



Durham E-Theses

A model for charge distribution in the gas and solid phases

Weaver, Malcolm B.

How to cite:

Weaver, Malcolm B. (1995) *A model for charge distribution in the gas and solid phases*, Durham theses, Durham University. Available at Durham E-Theses Online: <http://etheses.dur.ac.uk/5311/>

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a [link](#) is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the [full Durham E-Theses policy](#) for further details.

A Model for Charge Distribution in the Gas and Solid Phases

Malcolm B. Weaver

Chemistry Department
University of Durham

M. Sc. Thesis

October 1995

The copyright of this thesis rests
with the author. No quotation
from it should be published
without the written consent of the
author and information derived
from it should be acknowledged.



09 MAY 1997

Malcolm B. Weaver

A Model for Charge Distribution
in the Gas and Solid Phases

M. Sc. Thesis

1995

University of Durham

M. Sc. Thesis

October 1995

A Model for Charge Distribution in the Gas and Solid Phases

Malcolm B. Weaver

Abstract

A technique is developed to model the change in molecular charge distribution as a system moves from the gas phase to the solid phase. The change in charge distribution is modelled by mixing occupied and unoccupied orbitals of the gas-phase system. The overall energy and charge distribution calculated is a function of a mixing parameter, θ . The overall energy is comprised of two terms, that due to the internal molecular energy and that due to the electrostatic interaction of charged species in the solid state, which stabilises the more ionic structures. The overall energy is calculated as a function of θ , and the lowest-energy arrangement is identified.

Initial calculations are made on the system in the gas phase, using semi-empirical Hartree-Fock theory. The results produced by this calculation are then input into the model, which calculates the energy and charge distribution for the solid phase. The model has been applied to the LiF system. The change in charge distribution for this system is not as large as anticipated and the reasons for this are discussed.

Acknowledgements

I would like to thank my supervisor Jeremy Hutson for help and guidance throughout this work. I would also like to thank my colleagues, Elizabeth Colbourn and John Kendrick, the first for getting me started and the second for making sure I finished.

I acknowledge the support of ICI, through partially payment of my fees and time and resources to work on this degree.

Finally, to Philippa, who kept me going through the difficult times.

Statement

No part of this thesis has previously been submitted by me for a degree to any university. Every effort has been made to ensure that all the work which is not original to the author has been properly credited. I place no restriction upon access to, or copying of, this thesis.

Contents

1. Introduction	2
1.1 Charges	3
1.2 Solid State Calculations	3
1.3 The Cluster Approach	4
1.4 Periodic Boundary Conditions	4
1.5 Pairwise Potentials	5
1.6 Summary	5
2. The Excited States	7
2.1 Orbital Energies and Integrals	7
2.2 State Energies	8
3. The Solid Phase	10
3.1 The Gas Phase System	10
3.2 The Solid Phase System	11
4. The Ewald Method	14
4.1 Introduction	14
4.2 Description	14
5. The Ewald Program	19
5.1 Value of ν	19
5.2 The Program	20
5.3 Results	21
6. The MOPAC program	23
6.1 MOPAC	23
6.2 Modifications to MOPAC	23
7. Verification	26
7.1 The Analyse Program	26
7.2 The Charges Program	29
7.3 Conclusions	30
8. Corresponding Orbitals	31
8.1 The Theory of Corresponding Orbitals	31
8.2 Implementation	33
9. The Solid Program	35

9.1	Application	35
10.	Results from the Solid Program	36
11.	Future Development	43
A.	Atomic Units and Fundamental Constants	44
B.	Program Listing of Ewald	45
C.	Ewald Program Data Set for NaCl	56
D.	Ewald Program Output for NaCl	57
E.	Mopac Subroutine solidorb.f	59
F.	.orb output file for LiF	63
G.	Program Listing of Analyse	66
H.	Analyse Output for Water by the AM1 Method	72
I.	Program Listing of Charges	73
J.	Charges Output for Water by the AM1 Method	83
K.	Program Listing of Corresp	86
L.	Program Listing of Solid	106
M.	Solid Output for LiF	134
12.	References	139
13.	Tables	143

PART I

Introduction



1: Introduction

The charge distribution of a molecule in the solid phase is different from that of the gas phase. This phenomenon is particularly pronounced for the hydrogen halides. In the gas phase the bonding of hydrogen halides, whilst very polar, contains considerable covalent character. Another example of this is the hydrogen fluoride / ammonia system which in the gas phase exists as discrete hydrogen fluoride and ammonia molecules but in the solid phase forms the ammonium fluoride species.

The aim of this work was to devise a technique that can model the change in the molecular electronic distribution as the chemical environment of a system goes from the gas phase to the solid or crystalline phase. We intend only to account for the change in charge distribution and energy for a molecule of fixed geometry. In the solid phase the Madelung effect, due to the electrostatic interaction of charged species, will stabilise the more ionic structures. We model the change in charge distribution and energy by mixing ground state occupied orbitals with ground state unoccupied orbitals. This is done with no change in the geometry of the system. We are making the basic assumption that all orbitals except the set of active orbitals, which is made up of occupied and unoccupied molecular orbitals, are invariant between the gas and solid phases. The validity of this assumption will be discussed later.

The total energy of the solid-state system is considered to be the sum of the molecular energy and the electrostatic energy

$$E_{\text{total}} = E_{\text{molecular}} + E_{\text{electrostatic}}$$

The solid-state energy will be a function of the degree of mixing within the set of active orbitals and will be minimised with respect to this mixing to give the lowest possible solid-state energy.

The method for calculating the molecular energy of the system has been split into two problems within our methodology. The first is to calculate the energies, and charge distributions, of the excited states from the state energy, orbital energies, and coulomb and exchange integrals of the ground state. The second is to combine these state energies and charge distributions for the gas phase to give the ground state energy and charge distribution for the solid phase. The electrostatic energy is that due to the energy of a set of

charges in a periodic lattice and is calculated by the Ewald method.

Throughout this thesis all quantities are in atomic units, unless otherwise stated. The values of the atomic units are in Appendix A.

1.1 Charges

There are many methods of obtaining atomic charges from electronic structure calculations. If we ignore any sort of multipole description, either distributed or otherwise, then the methods fall into two categories. The first category is composed of methods that partition the orbital population into atomic charges, such as those of Mulliken,¹ Politzer *et al.*,² Jug,³ Bader *et al.*⁴ and Jug.⁵ These charge methods tend to be assigned a certain amount of physical meaning and consequently were originally the only type of charges used. The second category are those that involve fitting charges to the electrostatic potential or wavefunction, such as those of Chirlian & Francl,⁶ Orozco & Luque,⁷ Tasi *et al.*,⁸ Rodriguez *et al.*,⁹ Su,¹⁰ Alemán *et al.*¹¹ and Momany.¹² Fitting methods usually produce charges that are significantly larger than those produced by partitioning methods and indeed larger than is acceptable in terms of electronic density on a particular atom. For this reason they are felt to be un-physical and are often criticised. However, these charges do not intend to represent the actual electronic density on an atom but the electrostatic potentials around the molecule, particularly at long distances (on an atomic scale) and, for this purpose, do a much better job than charges calculated from partitioning methods.

1.2 Solid State Calculations

As with isolated molecules, it is possible to model the solid state at a variety of levels of accuracy. The level one chooses will be influenced by several factors. These factors will include the particular features of the system one is trying to investigate and how computationally expensive a method is, taking into account the size of the system under investigation. We are interested in electronic effects, namely the change in charge distribution between the gas and solid phase, and therefore will have to use relatively high levels of accuracy to address this. We are interested in electronic structure calculation, which is the domain of *ab initio* and semi-empirical calculations. Pairwise potential and molecular mechanics methods can also be used to model solids, as with isolated molecules, but they do not contain the information about the charge distribution that we need. Also we are, at present, considering

only inorganic solids in our studies. Studies of organic solids would also be amenable to most of the techniques described,¹³ but metals require very different techniques.¹⁴

There are two approaches to modelling solids. In the first a cluster of atoms represents the solid and in the second periodic boundary conditions are applied to a unit cell.

1.3 The Cluster Approach

Cluster methods are well established.¹⁵ Standard quantum-mechanical (QM) techniques can be used to study the clusters. These can give electronic density distributions, charges and other aspects of the bonding. By varying the geometry of the cluster, interatomic potentials and force constants may be deduced.

For cluster calculations, there are three choices to be made. The first is the size of the cluster. The larger the cluster, the better it will represent the solid but the more computationally expensive the calculation will be. The second consideration is the termination of the solid. For a system such as a zeolite it is not possible to terminate the crystal without leaving some bonds dangling. These can be saturated with a hydrogen atom or closed-shell ion, depending on the system in question. For an organic solid it is simply necessary to ensure that the cluster is composed of whole molecules. Lastly, one has to decide on the QM method to be used. This will depend on the size of system that one wishes to investigate and the particular features of the system that are of interest.

1.4 Periodic Boundary Conditions

Periodic Boundary Conditions (PBC) represent the crystal as an infinite array of unit cells. If one wishes to investigate defect features then a supercell approach is used, where the defect is made in one unit cell which is then surrounded by perfect unit cells to form the supercell. This is important to isolate the defects from one another. One of the most important methodologies on the modelling of defects is that of Mott & Littleton.¹⁶

At the QM level PBC can be addressed by both Hartree-Fock (HF) and Density Functional Theory (DFT) methods. Work using HF methodology is exemplified by the work of Pisani *et al.*,¹⁷ using the CRYSTAL code.

1.5 Pairwise Potentials

Pairwise potential methods have proved very successful in modelling ionic materials, such as metal oxide systems. These are exemplified by the HADES/PLUTO¹⁸ codes. These methods include some electronic effects via a core/shell model which allows for polarisation. However, the effect we are modelling is that of charge transfer and that cannot be easily included using pair potentials.

1.6 Summary

We are modelling charge transfer effects and therefore require QM calculations. We chose **MOPAC** to do the initial calculations on which our method is based because the program is generally available and the theory is well documented. The equations describing the state energy and expectation values between determinants are standard.¹⁹ In practice any of the charge methods described in the first category could be used to calculate the charges. However, because of its simplicity the Mulliken method for calculating the charge distribution has been adopted. This is also the method used by **MOPAC**.

PART II

Theory

2: The Excited States

In our model we describe the chemical system using restricted Hartree-Fock (RHF) molecular orbital theory as opposed to unrestricted Hartree-Fock (UHF) molecular orbital theory. In UHF molecular orbital theory each spin orbital function is a spatial orbital function multiplied by a term for the spin of the electron, i.e.

$$\zeta(i) \equiv \zeta(x_i, y_i, z_i, \xi_i) = \psi_i(x_i, y_i, z_i)\alpha(\xi_i),$$

where ζ is a spin orbital, ψ_i is a spatial orbital and α is a spin function. In RHF the approximation is made that the spatial component of the α and β spin orbitals are the same. The spin orbital associated with the electron of α spin is denoted ψ_i , and the complementary orbital (for β spin) is denoted $\bar{\psi}_i$.

2.1 Orbital Energies and Integrals

In molecular orbital theory the Hamiltonian is of the form

$$H = T_N + T_e + V_{eN} + V_{ee} + V_{NN},$$

where T_N is the kinetic energy of the nuclei, T_e the kinetic energy of the electrons, V_{eN} the electron - nuclear attractive Coulomb potential, V_{ee} the electron repulsive Coulomb potential and V_{NN} the nuclear repulsive Coulomb potential.

Considering a closed-shell wavefunction, for a system with N electrons of the form

$$|\Psi_0\rangle = |\zeta_1 \cdots \zeta_a \cdots \zeta_N\rangle$$

or equivalent, emphasising the RHF nature of the wavefunction

$$|\Psi_0\rangle = |\psi_1 \bar{\psi}_1 \cdots \psi_a \bar{\psi}_a \cdots \psi_{N/2} \bar{\psi}_{N/2}\rangle$$

formulae can be written for the one- and two-electron integrals and the orbital and state energies.

The one-electron integrals, h_{ij} , are the matrix elements of $h(\mathbf{r}_1)$, which is the core Hamiltonian describing an electron's kinetic energy and potential energy in the field of the nuclei. These are given by²⁰

$$(i|h|j) = h_{ij} = \int \psi_i^*(\mathbf{r}_1)h(\mathbf{r}_1)\psi_j(\mathbf{r}_1)d\mathbf{r}_1.$$

The coulomb and exchange integrals are two special forms of the generic two-electron integral,²⁰

$$(ij|kl) = \int \psi_i^*(\mathbf{r}_1)\psi_j(\mathbf{r}_1)\frac{1}{r_{12}}\psi_k^*(\mathbf{r}_2)\psi_l(\mathbf{r}_2)d\mathbf{r}_1d\mathbf{r}_2.$$

The coulomb integral is defined as

$$\mathcal{J}_{ij} = (ii|jj)$$

and the exchange integral is defined as

$$\mathcal{K}_{ij} = (ij|ji).$$

The orbital energy for an occupied orbital, ε_i , within the RHF approximation, can be expressed in terms of these one-electron and two-electron integrals,²¹

$$\varepsilon_i = h_{ii} + \sum_a^{N/2} (2\mathcal{J}_{ia} - \mathcal{K}_{ia}), \quad (1)$$

where a spans the occupied orbitals and N is the number of electrons in the system. This energy includes the kinetic energy and attraction to the nuclei, together with coulomb and exchange interactions with the electrons in the other orbitals. This equation, as shown in the form above, is only valid for systems where all the orbitals are doubly occupied.

2.2 State Energies

It is possible to describe the ground-state energy of a closed-shell system, S_0 , by a combination of the orbital energies and integrals,²²

$$E_0 = 2 \sum_a h_{aa} + \sum_a \sum_b (2\mathcal{J}_{ab} - \mathcal{K}_{ab}), \quad (2)$$

2: The Excited States

where a and b span the occupied orbitals.

If electrons are promoted from an active occupied molecular orbital (AOMO), denoted by s , to an active unoccupied molecular orbital (AUMO), denoted by u , it is possible to define two new singlet states; a singly excited singlet, S_1 , and a doubly excited singlet, S_2 . The energies of S_1 and S_2 relative to S_0 are²³

$$E_1 - E_0 = \varepsilon_u - \varepsilon_s - \mathcal{J}_{su} + 2\mathcal{K}_{su} \quad (3)$$

and

$$E_2 - E_0 = 2\varepsilon_u - 2\varepsilon_s + \mathcal{J}_{ss} + \mathcal{J}_{uu} - 4\mathcal{J}_{su} + 2\mathcal{K}_{su}. \quad (4)$$

For the S_1 and S_2 states, equations analogous to (2) are

$$\begin{aligned} E_1 = & \sum_{a \neq s}^N (2h_{aa}) + h_{ss} + h_{uu} + \sum_{a \neq s}^N \sum_{b \neq s}^N (2\mathcal{J}_{ab} - \mathcal{K}_{ab}) \\ & + \sum_{a \neq s}^N (2\mathcal{J}_{ua} + 2\mathcal{J}_{sa} - \mathcal{K}_{ua} - \mathcal{K}_{sa}) + \mathcal{J}_{su} + \mathcal{K}_{su} \end{aligned}$$

and

$$\begin{aligned} E_2 = & \sum_{a \neq s}^N (2h_{aa}) + 2h_{uu} + \sum_{a \neq s}^N \sum_{b \neq s}^N (2\mathcal{J}_{ab} - \mathcal{K}_{ab}) \\ & + \sum_{a \neq s}^N (4\mathcal{J}_{ua} - 2\mathcal{K}_{ua}) + \mathcal{J}_{uu}. \end{aligned}$$

2: The Excited States

3: The Solid Phase

We wish to calculate the effect of a change of environment on our system and this is modelled by mixing occupied and unoccupied orbitals. We are attempting to calculate the energy and charge distribution by combining the orbital energies, coulomb and exchange integrals, calculated for the ground state of the isolated molecule. We are then going to use the energies of the states and the charge distribution to calculate the molecular energy of the solid phase. This is then combined with the electrostatic energy to give the total energy of the system in the solid phase. The theory for calculating the molecular energy, once we have the state energies for the gas phase, is described in this chapter. Firstly, we need to describe all three states accurately in the gas phase in terms of molecular orbital theory.

3.1 The Gas Phase System

First of all some notation needs to be described. A molecular orbital, which in the theory we are using is a linear combination of atomic orbitals, is denoted ψ_i for the gas phase or χ_i for the solid phase. A many-electron wavefunction, denoted Ψ or X respectively, is made up of the anti-symmetrised product of a set of molecular orbitals. This is usually done using determinants.

$$\Psi = (n!)^{-1/2} \begin{vmatrix} \zeta_1(1) & \zeta_2(1) & \dots & \zeta_n(1) \\ \zeta_1(2) & \zeta_2(2) & \dots & \zeta_n(2) \\ \vdots & \vdots & & \vdots \\ \zeta_1(n) & \zeta_2(n) & \dots & \zeta_n(n) \end{vmatrix}.$$

In the present version of our model we consider just two orbitals in the active set of orbitals, the occupied active orbital (AOMO) and the unoccupied active orbital (AUMO). The general concept of mixing orbitals does not restrict us to just two orbitals, but all the mathematics developed in the remainder of this thesis is based on this.

The gas-phase states that we use within our model are the ground state (S_0) and two excited singlet states, one formed by the single excitation of an electron from the AOMO to the AUMO (S_1) and the other by the double excitation of both electrons from the AOMO to the AUMO (S_2). The AOMO orbital is represented by ψ_0 and the AUMO orbital by ψ_1 .

3: The Solid Phase

In the gas phase these three states are described by the following wavefunctions,

$$\begin{aligned} S_0 & \Psi_0 = |\text{core } \psi_0 \bar{\psi}_0|, \\ S_1 & \Psi_1 = \frac{1}{\sqrt{2}} (|\text{core } \psi_0 \bar{\psi}_1| - |\text{core } \bar{\psi}_0 \psi_1|), \\ S_2 & \Psi_2 = |\text{core } \psi_1 \bar{\psi}_1|. \end{aligned}$$

In our methodology the active orbitals are the AOMO and AUMO and the core orbitals are all the other occupied molecular orbitals.

3.2 The Solid Phase System

In the solid phase the three states are described by the following wavefunctions,

$$\begin{aligned} S_0 & X_0 = |\text{core } \chi_0 \bar{\chi}_0|, \\ S_1 & X_1 = \frac{1}{\sqrt{2}} (|\text{core } \chi_0 \bar{\chi}_1| - |\text{core } \bar{\chi}_0 \chi_1|), \\ S_2 & X_2 = |\text{core } \chi_1 \bar{\chi}_1|. \end{aligned}$$

χ_0 and χ_1 can loosely be described as the AOMO and AUMO of the system in the solid phase. They can be defined by the following two equations.

$$\chi_0 = \cos \theta \psi_0 + \sin \theta \psi_1$$

and

$$\chi_1 = -\sin \theta \psi_0 + \cos \theta \psi_1,$$

where θ is a variable that describes the degree of mixing.

In the solid phase we are interested mostly in calculating the ground state energy. The ground state wavefunction in the solid phase may be expanded,

$$\begin{aligned} X_0 &= |\text{core } \chi_0 \bar{\chi}_0| \\ &= |\text{core } (\cos \theta \psi_0 + \sin \theta \psi_1) (\overline{\cos \theta \psi_0 + \sin \theta \psi_1})| \\ &= \cos^2 \theta |\text{core } \psi_0^2| + \sin^2 \theta |\text{core } \psi_1^2| + \cos \theta \sin \theta (|\text{core } \psi_0 \bar{\psi}_1| + |\text{core } \psi_1 \bar{\psi}_0|) \end{aligned}$$

3: The Solid Phase

Using the definitions of Ψ_0 , Ψ_1 and Ψ_2 given in the previous section and the fact that $|\text{core } \psi_1 \bar{\psi}_0| = -|\text{core } \bar{\psi}_0 \psi_1|$ we get

$$X_0 = \cos^2 \theta \Psi_0 + \sin^2 \theta \Psi_2 + \sqrt{2} \sin \theta \cos \theta \Psi_1.$$

The three wavefunctions Ψ_0 , Ψ_1 and Ψ_2 are the wavefunctions corresponding to the ground state, the singly excited singlet and the doubly excited singlet state in the gas phase. The corresponding molecular energies of these states are the expectation values of the intramolecular Hamiltonian.

$$\text{For } S_0, E_0 = \langle \Psi_0 | \mathcal{H} | \Psi_0 \rangle.$$

$$\text{For } S_1, E_1 = \langle \Psi_1 | \mathcal{H} | \Psi_1 \rangle.$$

$$\text{For } S_2, E_2 = \langle \Psi_2 | \mathcal{H} | \Psi_2 \rangle.$$

The intramolecular Hamiltonian expectation value of the ground state in the solid phase, $\langle X_0 | \mathcal{H} | X_0 \rangle$, is

$$\begin{aligned} & \cos^4 \theta \langle \Psi_0 | \mathcal{H} | \Psi_0 \rangle + \cos^2 \theta \sin^2 \theta \langle \Psi_0 | \mathcal{H} | \Psi_2 \rangle + \sqrt{2} \cos^3 \theta \sin \theta \langle \Psi_0 | \mathcal{H} | \Psi_1 \rangle \\ & + \cos^2 \theta \sin^2 \theta \langle \Psi_2 | \mathcal{H} | \Psi_0 \rangle + \sin^4 \theta \langle \Psi_2 | \mathcal{H} | \Psi_2 \rangle + \sqrt{2} \cos \theta \sin^3 \theta \langle \Psi_2 | \mathcal{H} | \Psi_1 \rangle \\ & + \sqrt{2} \cos^3 \theta \sin \theta \langle \Psi_1 | \mathcal{H} | \Psi_0 \rangle + \sqrt{2} \cos \theta \sin^3 \theta \langle \Psi_1 | \mathcal{H} | \Psi_2 \rangle \\ & + 2 \cos^2 \theta \sin^2 \theta \langle \Psi_1 | \mathcal{H} | \Psi_1 \rangle, \end{aligned}$$

which can be reduced to

$$\begin{aligned} & \cos^4 \theta \langle \Psi_0 | \mathcal{H} | \Psi_0 \rangle + \sin^4 \theta \langle \Psi_2 | \mathcal{H} | \Psi_2 \rangle + 2 \cos^2 \theta \sin^2 \theta \langle \Psi_1 | \mathcal{H} | \Psi_1 \rangle \\ & + 2 \cos^2 \theta \sin^2 \theta \langle \Psi_0 | \mathcal{H} | \Psi_2 \rangle + 2\sqrt{2} \cos \theta \sin^3 \theta \langle \Psi_1 | \mathcal{H} | \Psi_2 \rangle. \end{aligned}$$

The values of $\langle \Psi_0 | \mathcal{H} | \Psi_0 \rangle$, $\langle \Psi_1 | \mathcal{H} | \Psi_1 \rangle$ and $\langle \Psi_2 | \mathcal{H} | \Psi_2 \rangle$ have been described previously. The value of $\langle \Psi_0 | \mathcal{H} | \Psi_1 \rangle$ is zero, due to Brillouin's Theorem, which states that a singly excited determinant will not directly interact with the ground state determinant.²⁴ The values of $\langle \Psi_0 | \mathcal{H} | \Psi_2 \rangle$ and $\langle \Psi_1 | \mathcal{H} | \Psi_2 \rangle$,

3: The Solid Phase

labelled E_{02} and E_{12} for ease of reference, can be calculated using rules for calculating matrix elements between determinants.²⁵ These give

$$\langle \Psi_0 | \mathcal{H} | \Psi_2 \rangle = \mathcal{K}_{su}$$

and

$$\langle \Psi_1 | \mathcal{H} | \Psi_2 \rangle = \sqrt{2} \left[h_{su} + \sum_{a \neq u}^N 2 (su | aa) + (su | uu) \right].$$

4: The Ewald Method

4.1 Introduction

To calculate the electrostatic energy of a set of point charges one simply uses the following equation,

$$E_{\text{Electrostatic}} = \sum_{ij} \frac{Z_i Z_j}{r_{ij}}$$

For a periodic system the equation is slightly modified,²⁶

$$E_{\text{Electrostatic}} = \frac{1}{2} \sum_{\mathbf{n}}' \left(\sum_{ij} \frac{Z_i Z_j}{|\mathbf{r}_{ij} + \mathbf{n}|} \right) \quad (6)$$

where $\mathbf{n} = (n_x L_x, n_y L_y, n_z L_z)$ with n_x , n_y and n_z integer and L_x , L_y and L_z are the unit cell lengths. The prime indicates that the original cell ($n_x = n_y = n_z = 0$) is not included in the summation. The inner sum runs over the lattice sites with the unit cell and the outer sum runs over unit cells.

This equation converges very slowly and so is very rarely used in practice. Ewald proposed a method of replacing the point charges with Gaussian charge distributions and transforming into a rapidly converging series.²⁷

4.2 Description

The Ewald sum method is one of the most efficient ways to calculate the electrostatic energy of a set of charges on an infinite lattice.²⁸ The calculation is divided between real space and reciprocal space. We can consider the total electrostatic potential, φ , at a point as the sum of two related potentials. The first potential, φ_1 , is for a Gaussian distribution of charge situated at each ion site (the signs are the same as those of the real ions). The Gaussian distribution of the charge situated at the reference point, which is the point at which the potential is situated, does not contribute to the potential, as the point charge it is representing is zero distance from itself. Therefore, φ_a is defined as the potential with all the charge distributions present, whereas φ_b is the potential due to only the charge distributions of the charge at the reference point. The potential that is needed is φ_1 which is the difference between φ_a and φ_b .

The second potential, φ_2 , is due to each ion being replaced by a Gaussian charge distribution and counterbalanced with a point charge equal and opposite to the ion. Thus we have two equations,

$$\varphi = \varphi_1 + \varphi_2,$$

and

$$\varphi_1 = \varphi_a - \varphi_b,$$

where φ is the total electrostatic potential, φ_1 is the potential with ions replaced by gaussians, excluding the point charge, φ_2 is the potential of point ions with gaussians of opposite charge, φ_a is the potential with ions replaced by gaussian, including gaussian at point charge of interest, φ_b is the potential due to gaussian at point charge of interest.

The reason for splitting the charge potential in this fashion is that, with the correct choice of the width of the gaussian peak, η , we can get rapid convergence of both potentials at the same time. The calculation of the best value for η is discussed in section 5.1. The choice of η has no effect on the value of the energy, it merely defines the partitioning of the calculation between the real and reciprocal spaces.

The potential of the infinite lattice of Gaussian distributions is described by

$$\varphi_a = \sum_{\mathbf{G}} c_{\mathbf{G}} e^{(i\mathbf{G} \cdot \mathbf{r})}, \quad (7)$$

and the corresponding charge distribution is described by

$$\rho_a = \sum_{\mathbf{G}} \rho_{\mathbf{G}} e^{(i\mathbf{G} \cdot \mathbf{r})}, \quad (8)$$

where \mathbf{r} is the vector from the reference point, \mathbf{G} is 2π times a vector in the reciprocal lattice and $c_{\mathbf{G}}$ and $\rho_{\mathbf{G}}$ are coefficients.

Substituting (7) and (8) into the Poisson equation,

$$\nabla^2 \varphi_a = -4\pi \rho_a,$$

which relates φ_a and ρ_a , we get

4: The Ewald Method

$$\sum G^2 c_{\mathbf{G}} e^{(i\mathbf{G}\cdot\mathbf{r})} = 4\pi \sum \rho_{\mathbf{G}} e^{(i\mathbf{G}\cdot\mathbf{r})},$$

which simplifies to

$$c_{\mathbf{G}} = \frac{4\pi}{G^2} \rho_{\mathbf{G}}. \quad (9)$$

Each point is the centre of a Gaussian charge distribution,

$$\rho(\mathbf{r}) = \sum_t q_t \left(\frac{\eta}{\pi}\right)^{3/2} e^{(-\eta r_c^2)},$$

where $\rho(\mathbf{r})$ is the charge distribution of a single gaussian, q_t is the charge of the corresponding point ion, η is the width of the gaussian, r_c is the distance from q_t .

Multiplying by $e^{(-i\mathbf{G}\cdot\mathbf{r})}$ and integrating over all space,

$$\begin{aligned} \rho_{\mathbf{G}} \int_{\text{one cell}} e^{(i\mathbf{G}\cdot\mathbf{r})} e^{(-i\mathbf{G}\cdot\mathbf{r})} d\mathbf{r} &= \rho_{\mathbf{G}} \Delta \\ &= \int_{\text{all space}} \sum_t q_t \left(\frac{\eta}{\pi}\right)^{3/2} e^{(-\eta(|\mathbf{r}-\mathbf{r}_t|)^2)} e^{(-i\mathbf{G}\cdot\mathbf{r})} d\mathbf{r}, \end{aligned}$$

where Δ is the volume of unit cell, \mathbf{r} is the absolute position vector, \mathbf{r}_t is the position vector of the ion relative to the unit cell origin. Thus

$$\begin{aligned} \rho_{\mathbf{G}} \Delta &= \sum_t q_t e^{(-i\mathbf{G}\cdot\mathbf{r}_t)} (\eta/\pi)^{3/2} \int_{\text{one cell}} e^{[-(i\mathbf{G}\cdot\xi + \eta\xi^2)]} d\xi \\ &= \left(\sum_t q_t e^{(-i\mathbf{G}\cdot\mathbf{r}_t)} \right) e^{(-G^2/4\eta)} = S(\mathbf{G}) e^{(-G^2/4\eta)}, \end{aligned}$$

where $S(\mathbf{G})$ is $\sum_t q_t e^{(-i\mathbf{G}\cdot\mathbf{r}_t)}$, the structure factor in appropriate units.

Combining (7) and (9) we get

$$\varphi_a = \frac{4\pi}{\Delta} S(\mathbf{G}) G^{-2} e^{(i\mathbf{G}\cdot\mathbf{r} - G^2/4\eta)}, \quad (10)$$

4: The Ewald Method

which at the origin, $r = 0$, reduces to

$$\varphi_a = \frac{4\pi}{\Delta} S(\mathbf{G}) G^{-2} e^{(-G^2/4\eta)}. \quad (11)$$

The potential φ_b due to the gaussian at the point of interest is

$$\varphi_b = \int_0^\infty (4\pi r^2 dr) (\rho/r) = 2q_i(\eta/\pi)^{1/2},$$

and so

$$\varphi_1(i) = \frac{4\pi}{\Delta} S(\mathbf{G}) G^{-2} e^{(-G^2/4\eta)} - 2q_i(\eta/\pi)^{1/2}.$$

The potential φ_2 is due to three contributions from each ion point,

$$\varphi_2 = q_l \left[\frac{1}{r_l} - \frac{1}{r_l} \int_0^{r_l} \rho(\mathbf{r}) d\mathbf{r} - \int_{r_l}^\infty \frac{\rho(\mathbf{r})}{r} d\mathbf{r} \right].$$

On substituting for $\rho(\mathbf{r})$ we have

$$\varphi_2 = \sum_l \frac{q_l}{r_l} F(\eta^{1/2} r_l), \quad (12)$$

where

$$F(x) = \left(\frac{2}{\sqrt{\pi}} \right) \int_x^\infty e^{-s^2} ds.$$

Finally,

$$\varphi(i) = \frac{4\pi}{\Delta} S(\mathbf{G}) G^{-2} e^{(-G^2/4\eta)} - 2q_i(\eta/\pi)^{1/2} + \sum_l \frac{q_l}{r_l} F(\sqrt{\eta} r_l). \quad (13)$$

PART III

Programs

5: The Ewald Program

5.1 Value of ν

There is a formula²⁹ to calculate the value of ν that will calculate the total energy in the most efficient way

$$\nu = \left(\frac{s\pi^3}{V_c^2} \right)^{1/6},$$

where s is the number of atoms in the unit cell and V_c is the volume of the unit cell. This formula comes from minimising the total number of terms in the Ewald calculation. ν is related to η by

$$\nu = \sqrt{\eta}.$$

Using the example of a cubic box with sides of unit length, we get

$$\nu = \left(\frac{8\pi^3}{16} \right)^{1/6},$$

which gives a value for ν of 2.506628275.

For NaCl, using atomic units

$$\nu = \left(\frac{8\pi^3}{(10.43129 \text{ Bohr})^6} \right)^{1/6},$$

this gives ν as 0.240299009 reciprocal Bohr.

The time taken to calculate the energy of the NaCl unit cell, and the energy itself, for various values of ν , is shown in Table 1. The value of ν is given using the cubic box length as the unit of length. In our implementation of the method the best value of ν is calculated for each data set, thus ensuring that the calculation is as fast as possible. In Table 1 κ_{\max} is the number of reciprocal unit cells with Vec the corresponding number of vectors generated. T_κ and $V^{zz}(\kappa)$ are the time taken and the energy value produced by the reciprocal space part of the program. Cells is the number of unit cells needed by the real space part of the program and T_R and $V^{zz}(R)$ the time taken

and the energy produced. T is the total time taken and V^{zz} is the total electrostatic energy produced. Timings are given in seconds and the energies are in hartrees per unit cell. The calculations were performed on a Sun 3/60. Table 1 shows that the calculation is quickest when ν is 2.5, which is the same, to 3 significant figures, as the predicted value of 2.506628275.

Table 1 Time taken to calculate the total energy for various values of ν

Results of the Ewald Program									
ν	κ_{\max}	Vec	T_{κ}	$V^{zz}(\kappa)$	Cells	T_R	$V^{zz}(R)$	T	V^{zz}
1.0	4	180	2.32	-4.5135	4	22.02	-9.4669	24.348	-13.9804
2.0	4	180	2.38	-9.0105	3	8.92	-4.9700	11.30	-13.9805
2.5	4	180	2.36	-11.0692	2	4.48	-2.9113	6.84	-13.9805
3.0	5	337	5.40	-12.5284	2	2.72	-1.4521	8.12	-13.9805
4.0	6	534	10.24	-13.7603	2	2.72	-0.2203	12.96	-13.9805
5.0	7	831	17.86	-13.9610	2	2.70	-0.0195	20.56	-13.9805
6.0	8	1208	29.08	-13.9794	2	2.68	-0.0010	31.76	-13.9805
7.0	9	1689	44.84	-13.9805	2	2.68	$<10^{-4}$	47.52	-13.9805
8.0	10	2330	67.30	-13.9806	1	0.48	$<10^{-6}$	67.78	-13.9806
9.0	11	3039	96.66	-13.9805	1	0.50	$<10^{-8}$	97.16	-13.9805
10.0	12	3852	133.72	-13.9805	1	0.46	$<10^{-10}$	134.18	-13.9805

5.2 The Program

The book, “Computer Simulation of Liquids” by Allen & Tildesley,³⁰ contains listings of useful programs, one of which is a program to calculate the electrostatic energy of a set of point charges within a cubic unit cell. After inspection of this program some code was written that calculated the electrostatic energy for a set of points in an arbitrary unit cell. A listing of this program is given in Appendix B.

The input for the program specifies the fractional position of each ion within the unit cell, as well as defining the unit cell itself. A specimen data set, for NaCl, is shown in Appendix C. The program initially calculates the value of ν and converts all the data into atomic units. It then proceeds to calculate the real and reciprocal space components of the energy. In each case it continues the calculation, with an increasing number of cells, until the value of the energy in that space has converged. Finally it calculates the Madelung constant and the smallest ion-ion distance. This is for ease of comparison with other results.

The output first has a repeat of the input data to ensure that the data has been read correctly. The energy and timings for each of the real and reciprocal space calculations are then reported as well as the total time and energy. A specimen output, for NaCl, is shown in Appendix D.

5.3 Results

The results of the program were checked using two different methods. Firstly, for NaCl, a program was written that calculated the electrostatic energy of set of point charges in a cubic unit cell using the simple coulombic potential for a periodic system described in equation (6). This program is very simple and therefore it is easy to ensure that it is error free. This method obviously takes much longer to calculate the energy than the Ewald method. The variables in the program are all in atomic units and therefore the energy calculated is in hartrees per unit cell. To convert this to kJ/mol it is multiplied by the factor,

$$\frac{N_A e^2}{4\pi n \epsilon_0 L},$$

where n is the number of asymmetric units in the unit cell and L is the length of the unit cell. For NaCl the value of n is 4 and the value of L is 5.64056 Å.

For NaCl this gives V^{zz} as -13.9758 hartrees per unit cell which is equivalent to 860612 J/mol.

Coulson³¹ gives the electrostatic energy in the NaCl crystal as 8.92 eV per NaCl. This is equivalent to 861 kJ/mol. This agreement is to be expected as both calculations are very accurate.

Secondly, comparisons were made with values in the literature for simple inorganic structures. Mostly the literature reported these energies as a

5: The Ewald Program

Madelung constant and the shortest ion-ion distance in the crystal. The energy is calculated by the equation.³²

$$V^{zz} = \frac{N_A \mathcal{A} e^2 Z_i e Z_j e}{4 \pi \epsilon_0 R_0},$$

where \mathcal{A} is the Madelung constant, R_0 is the shortest ion-ion distance, Z_i and Z_j are the charges on the ions.

A comparison between the values calculated by the **Ewald** program and literature values for a variety of crystal structures is given in Table 2. The values of R_0 are calculated from the values of a_0 given in the references. The Madelung constants are from Weast.³³ The coordinates for the last three structures are found in Galasso.³⁴

Table 2 Comparison of Lattice Energies

Comparison of Lattice Energies					
	Literature			Ewald	
	$R_0/\text{\AA}$	\mathcal{A}	V^{zz}/Jmol^{-1}	\mathcal{A}	V^{zz}/Jmol^{-1}
NaCl	2.82028 ³⁵	1.74756	−860903	1.747564	−860903
CsCl	3.571 ³⁶	1.76267	−685797	1.762674	−685869
Wurtzite	2.3475 ³⁷	1.64132	−3885623	1.641307	−3885627
CaF ₂	2.36553 ³⁸	2.51939	−2959443	2.519393	−2959451
Rutile	1.9486 ³⁹	2.408	−13735256	2.383179	−13593665

Our value for Rutile differs from some values given in the literature. There seems to be some disagreement in the literature over the value of the Madelung constant for Rutile. The value given in the table above is from Weast,³³ whereas Moore⁴⁰ gives 4.7701 per unit cell, which is 2.38505 per TiO₂ unit.

6: The MOPAC program

6.1 MOPAC

MOPAC is a semi-empirical molecular orbital program and as such is parameterised. The three methods contained in **MOPAC 5** are MNDO⁴¹ (Modified Neglect of Differential Overlap), AM1⁴² (Austin Model 1) and PM3⁴³ (Parametric Method Number 3). Each method contains the full set of parameters needed for all the systems we studied, except for the atom Lithium, which can only be dealt with by MNDO (in **MOPAC 5**). AM1 and PM3 are types of MNDO calculations with different parameter sets. To avoid confusion the term MNDO will not be further used to refer to these two parameter sets, but only to the original MNDO method. **MOPAC** uses the minimal basis approach, which only includes the valence electrons, in which a hydrogen atom is described using one atomic orbital and each second row element is described by an s and three p atomic orbitals. All the systems we investigated contained only hydrogen and/or second row elements. It was unknown which of the three sets of parameters would give the best results for the work we were undertaking and so extensive calculations were carried out for each method.

6.2 Modifications to MOPAC

The program **MOPAC**⁴⁴ was used for all calculations performed. An additional subroutine was added to the program to make it calculate and write out the extra information that was needed by the **Solid** program, which is described later. This subroutine, **solidorb.f**, is listed in Appendix E. Minor modifications were also made to the subroutines **wrtkey.f** and **mndo.f** to allow the program to recognise the keyword **solidorb**, used to activate the extra portion of code, and to run that code. The additional code is designed only to be used when a CI calculation is performed, although there is no check to ensure that this happens. This was done because it is only during a CI calculation that **MOPAC** calculates the necessary integrals that are required for our method.

These additions caused **MOPAC** to write out two additional files, given the extensions **.bin** and **.orb**. Both files contain the same information, but **.bin** is a binary unformatted file, thereby retaining full numerical accuracy,

for use by the program **Solid**, whilst **.orb** is an ascii file purely for human consumption. A sample **.orb** file for LiF is listed in Appendix F. These files contain the one-electron integrals (h_{ij}), two-electron integrals (\mathcal{J} and \mathcal{K}) information, the energy (ϵ_i) and RHF molecular orbital coefficients and information relating the atomic orbitals to the relevant atoms to ensure that subsequent analysis programs correctly assign the atomic and molecular orbitals.

PART IV

Verification

7: Verification

Before we started using our methodology to calculate the energy and charge distribution of the solid phase we investigated several aspects of the initial **MOPAC** calculations that provide the data for our **Solid** program. Three aspects were investigated. The first was which of the three parameter sets would be used, MNDO, AM1 or PM3, the second was how well **MOPAC** was able to reproduce excited state energies and charge distributions, as compared to *ab initio* or experiment, and lastly what were the consequences of the state of the initial **MOPAC** calculation, i.e. whether the initial **MOPAC** calculation is for the ground state or for an excited state. The program **Analyse** was written to compare the energies and the program **Charges** the charges.

7.1 The **Analyse** Program

The program **Analyse** analyses the results produced by a series of **MOPAC** runs. A listing is given in Appendix G. The program will analyse all the **MOPAC** output files and the file created by our modification, for a given system and **MOPAC** method. In this way it is possible to compare the difference between the results calculated by our method and those of **MOPAC** or the literature, for a given system.

For each system, and within each parameter set, calculations were performed for four cases. The first calculation is for the ground state with the SCF (Self-Consistent Field) approximation. The next three calculations are done with a CI level of 2 (2 molecular orbitals are spanned by the configuration interaction); one calculation for the ground state, one for the first excited singlet state and one for the first excited triplet state. Apart from the differences described above, all the **MOPAC** calculations were done under the same conditions, using the keyword **nointer**, to stop the output of information that is not needed. All the calculations were done using geometries from the literature without allowing the program to minimise the structure. *Ab initio* or experimental structures from the literature were used as they are expected to be more accurate than those calculated by **MOPAC**. This also allowed better comparison with the energetic information from the literature.

The systems investigated and the sources of the geometry are shown in Table 3.

Table 3 Systems Investigated

Structure	Geometry
Carbon Monoxide	Experimental ⁴⁵
Furan	Experimental (Microwave spectra) ⁴⁶
Lithium Hydride	Ab-Initio (Multi CI) ⁴⁷
Hydrogen	Experimental ⁴⁸
Methane	Experimental ⁴⁹
Water	Ab-Initio (CISD) ⁵⁰

Table 4 shows the excitation energies that were available from the literature for our structures.

Table 4 Excitation Energies in the literature (in eV)

Structure	Singlet	Triplet
Carbon Monoxide ⁴⁵	8.51 ⁵¹	6.32 ⁵¹
Furan ⁵²		4.02 ⁵²
Lithium Hydride ⁴⁵	3.28540 ⁴⁵	
Hydrogen		
Methane		
Water ⁵³	8.6962 ⁵³	7.9011 ⁵³

Analyse initially shows the ground state energy for the system and the first singlet and first triplet excitation energies from the literature, if available. These excitation energies are read from a file and used as the basis for the comparison of the **MOPAC** results with experiment. The rest of the output shows calculations done with a CI level of 2.

The output for water using the AM1 method is shown in Appendix H. There are four columns of data headed 'MOPAC', 'Ground', 'Singlet' and 'Triplet'. The first column contains the energies from the **MOPAC** calculations. The other three columns contain the energies calculated by our method using ε_i , \mathcal{J} and \mathcal{K} , from a particular **MOPAC** calculation. The different columns are calculating the same information, e.g. 'Ground state energy' or 'Singlet state energy', using different **MOPAC** calculations, within our methodology. The first three rows of data show the energies of the three states, either from the **MOPAC** outputs or calculated by our method. The next two rows show the excitation energy calculated from the state energies. The following four rows show the error of the excitation energies with respect to the **MOPAC** calculations and the experimental excitations.

The data are summarised for all the structures investigated in Table 5. This shows the excitation energies (in eV) as calculated by both **MOPAC** and our method. The energies are all for the AM1 parameter set, except for Lithium Hydride which is for the MNDO parameter set.

Table 5 Excitation Energies Calculated by **MOPAC** (in eV)

Structure	MOPAC		This Work	
	Singlet	Triplet	Singlet	Triplet
Carbon Monoxide	6.53	4.82	6.14	4.50
Furan	4.17	2.68	4.87	2.51
Lithium Hydride	3.80	3.27	3.58	3.07
Hydrogen	9.49	7.34	9.43	7.27
Methane	9.13	7.63	9.27	7.66
Water	6.28	5.31	6.13	5.13

7.2 The Charges Program

The **Charges** program basically calculates charges for each of four data sets. Like the **Analyse** program, **Charges** operates on the data for one system and for a specific **MOPAC** method. The four data sets are the ground and excited states, calculated to a CI level of 2. For each data set the program calculates 4 sets of atomic charges. The ground state charge distribution is calculated from **MOPAC** orbitals optimised for the ground state and from **MOPAC** orbitals optimised for the excited state. Similarly the excited state charge distribution is calculated from **MOPAC** orbitals optimised for the ground state and from **MOPAC** orbitals optimised for the excited state. One of these charge distributions, for each state, uses the same orbitals that **MOPAC** uses and so agrees, to the accuracy of the program, with the charges reported by **MOPAC**. This is labelled "own MOs". The other charge distribution, for each state, uses the orbitals that are optimised for the other state. This is labelled "other MOs" and the agreement of this with the "own MOs" case, which we know is correct, is our measure of accuracy.

The charges are calculated, both within **MOPAC** and by **Charges**, by the Mulliken¹ method which sums the squares of the orbital coefficients of the orbitals on each atom. This gives the atomic charge. It is possible with this method to calculate the charge on an atom due to a particular molecular orbital. The **Charges** program actually splits the charge on each atom into that due to the core orbitals and that due to the active orbitals.

Charges calculates the atomic charges for the "own MOs" case and then the atomic charges for the "other MOs" case and compares the two. In each case the occupancy of the atomic orbitals and the corresponding atomic charges are reported for the core orbitals, the active orbitals and over all orbitals. Finally the errors between the "own MOs" atomic charges and the "other MOs" atomic charges are reported for the core orbitals, the active orbitals and over all orbitals. The error is calculated as the difference between the correct charge and the calculated charge, divided by the correct charge and expressed as a percentage.

A listing of the program is given in Appendix I and an output, for water, is given in Appendix J.

7.3 Conclusions

It was found that the parameter set had very little real effect on the results and so MNDO was chosen for the remainder of the calculations in this thesis, mainly because it is the only parameter set to include Lithium. The state chosen for the initial **MOPAC** calculations also had very little effect. The charges for a state using data calculated from a **MOPAC** calculation on that state were in exact agreement with the charges out of **MOPAC**. No set of orbitals from one state proved able to reproduce the charges of the other two states very accurately. An identical situation was found for the energies. For simplicity, all subsequent calculations were therefore based on orbitals calculated for the ground state. The energies calculated by our method are in agreement with those calculated directly by **MOPAC**. The agreement with experimental and *ab initio* results is not good but this is to be expected from a semi-empirical method.

8: Corresponding Orbitals

The **Analyse** program showed that our method was able to reproduce both the ground-state and the excited-state energies quite well using orbitals that had not been optimised for that state, within the **MOPAC** methodology.

However the reproduced charges are very poor and this is worthy of some discussion. Due to the variation principle an error in the wavefunction will produce a first order error in the charges but only a second order error in the energy. Consequently the charges produced using orbitals not optimised for a particular state will usually be poorer than the corresponding energy. Also the difference between the excited-state and ground-state charge distributions was initially expected to be due to a rearrangement of the electrons in the active orbitals. After study of the output of **Charges** it can be seen that the majority of the discrepancy between the calculated charges and the **MOPAC** charges is in fact due to the core charges, that portion of the charge that comes from the core orbitals. Therefore the change in charge distribution when going from one state to another cannot be described just by a movement of active electrons. We believe this is a real physical effect and not dependent on the method used to calculate the charges.

The theory of corresponding orbitals was investigated as it was hoped that this would determine a set of frozen core orbitals that would be invariant between the ground state and the excited singlet state.

The method is based upon work by King *et al.*⁵⁴ The method transforms two sets of orbitals, in this case those of the ground state and excited singlet state, so that their overlap matrix is diagonal. The transformation is unitary and therefore does not affect the state energy of either of the two states. The orbital coefficients and orbital energies of the two states are however modified. The method also assumes that the orbitals have been calculated using UHF theory. In our case, as they had been calculated by RHF, the orbitals coefficients for the α and β spins were initially the same. However, at the output of the program the α and β coefficients are no longer equal.

8.1 The Theory of Corresponding Orbitals

The theory and method of corresponding orbitals was obtained from the paper by King *et al.*⁵² It is based upon the idea that for two sets of spin orbitals it is possible to find two transformations that will create two new

sets of spin orbitals with maximum overlap.

Let Ψ_a and Ψ_b be single determinants

$$\Psi_a = |a_1(1)a_2(2) \cdots a_N(N)|,$$

$$\Psi_b = |b_1(1)b_2(2) \cdots b_N(N)|.$$

The orthogonalised spin-orbital sets are

$$\mathbf{a} = (a_1, a_2, \cdots, a_N),$$

$$\mathbf{b} = (b_1, b_2, \cdots, b_N).$$

The overlap matrix, between the spin-orbital set is \mathbf{D}

$$\mathbf{D} = \int \mathbf{b}^\dagger \mathbf{a} d\tau$$

There are two unitary transformations

$$\hat{\mathbf{a}} = \mathbf{a}\mathbf{V}$$

and

$$\hat{\mathbf{b}} = \mathbf{b}\mathbf{U}$$

which leave Ψ_a and Ψ_b invariant except in phase

$$\Psi_a = \det(\mathbf{V}^\dagger) |\hat{a}_1(1)\hat{a}_2(2) \cdots \hat{a}_N(N)|,$$

$$\Psi_b = \det(\mathbf{U}^\dagger) |\hat{b}_1(1)\hat{b}_2(2) \cdots \hat{b}_N(N)|.$$

that will transform the spin-orbital overlap matrix in the following manner

$$\hat{\mathbf{D}} = \int \hat{\mathbf{b}}^\dagger \hat{\mathbf{a}} d\tau = \mathbf{U}^\dagger \mathbf{D} \mathbf{V}.$$

8: Corresponding Orbitals

\mathbf{V} is the set of orthonormal eigenvectors of $\mathbf{D}^\dagger\mathbf{D}$

$$\mathbf{D}^\dagger\mathbf{D}\mathbf{V} = \mathbf{V}\mathbf{\Lambda},$$

$\mathbf{\Lambda}$ is diagonal and its elements, λ_i , are non-negative. If the λ_i are all non-zero the diagonal matrix $\mathbf{\Lambda}^{-1/2}$ can be constructed and used to make \mathbf{U}

$$\mathbf{U} = \mathbf{D}\mathbf{V}\mathbf{\Lambda}^{-1/2}.$$

The columns of \mathbf{V} are ordered such that $\lambda_i \geq \lambda_{i+1}$.

The number of non-zero eigenvalues is equal to the rank r of $\mathbf{D}^\dagger\mathbf{D}$. It becomes impossible to construct $\mathbf{\Lambda}^{-1/2}$ when r is less than the dimensionality N . It is still possible however to construct \mathbf{U} . Let the first r columns of \mathbf{U} be constructed as before

$$\mathbf{u}_i = \lambda_i^{-1/2}\mathbf{D}\mathbf{v}_i, \quad i \leq r.$$

The last $N - R$ columns \mathbf{U} can be any vectors provided that they are orthonormal to the first r columns.

8.2 Implementation

The corresponding orbital theory was coded up into a program called **Corresp**. This took the output of the **MOPAC** program, the **.bin** file, and performed the transformation on the orbitals contained within and produces a new set of orbitals in exactly the same format. These can be used in the same fashion as the those in the **.bin** file. The listing for **Corresp** is in Appendix K.

The corresponding orbital method had almost no effect on the atomic charges (the energy of course remains invariant to the corresponding orbital transformation). It was felt that this was because the orbitals produced by **MOPAC** for the systems considered here were all predominantly made up of a single atomic orbital. It is unknown whether other programs, perhaps *ab initio* ones, would show this feature and therefore the corresponding orbital treatment would have a more profound effect.

PART V

The Overall Technique

9: The Solid Program

The **Solid** program implements the theory described in the Solid Phase Chapter. It takes the output of the modified **MOPAC** program and a parameter θ , which describes the degree of mixing between the gas phase AOMO and AUMO orbitals in the solid phase ground state. A listing for **Solid** is in Appendix L.

9.1 Application

A sample **Solid** output for LiF is in Appendix M. The original **MOPAC** calculations were made using the MNDO parameter set. The program reads in the data from the **.orb** file and echoes it to ensure that it has been read in correctly. The program then goes on to calculate the orbital energies (to check against the ones read in) and the matrix elements between states, ready to input into the equation for the molecular energy of the solid phase,

$$E_{\text{molecular}} = \cos^4 \theta E_0 + \sin^4 \theta E_2 + 2 \cos^2 \theta \sin^2 \theta E_1 + 2 \cos^2 \theta \sin^2 \theta E_{02} \\ + 2\sqrt{2} \cos \theta \sin^3 \theta E_{12}$$

as a function of the mixing parameter θ .

The program then calculates the charges from the molecular orbitals. The charges can be verified to be correct by comparing them with **MOPAC** calculation.

The program then constructs the molecular orbital coefficients for the solid phase by rotating between the ψ_0 and ψ_1 orbitals by θ to form the new orbitals χ_0 and χ_1 . This set of orbitals then undergoes exactly the same operations to produce charges for the solid phase.

Finally the energy of the solid phase is calculated using the equation shown above.

The program also outputs a file suitable for input to the **Ewald** program, containing the charges of the solid phase ground state.

10: Results from the Solid Program

As described in the Introduction we are trying to minimise the solid-state energy of the system under investigation. It was originally envisioned that the two orbitals we would mix would be the HOMO and LUMO. For LiF, within the **MOPAC** methodology, the HOMO is orbital 4 and the LUMO is orbital 5. As can be seen from the portion of the **Solid** output shown below,

Orbital co-efficients (column vectors)

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
.14884	-.15960	.00000	.00000	-.87303	.00000	.00000	-.43611
.21924	-.15254	.00000	.00000	.48209	.00000	.00000	-.83442
.00000	.00000	-.17173	.00355	.00000	-.98493	-.02035	.00000
.00000	.00000	.00355	.17173	.00000	.02035	-.98493	.00000
.96312	.10632	.00000	.00000	.02177	.00000	.00000	.24620
-.04662	.96951	.00000	.00000	-.07025	.00000	.00000	-.23008
.00000	.00000	-.98493	.02035	.00000	.17173	.00355	.00000
.00000	.00000	.02035	.98493	.00000	-.00355	.17173	.00000

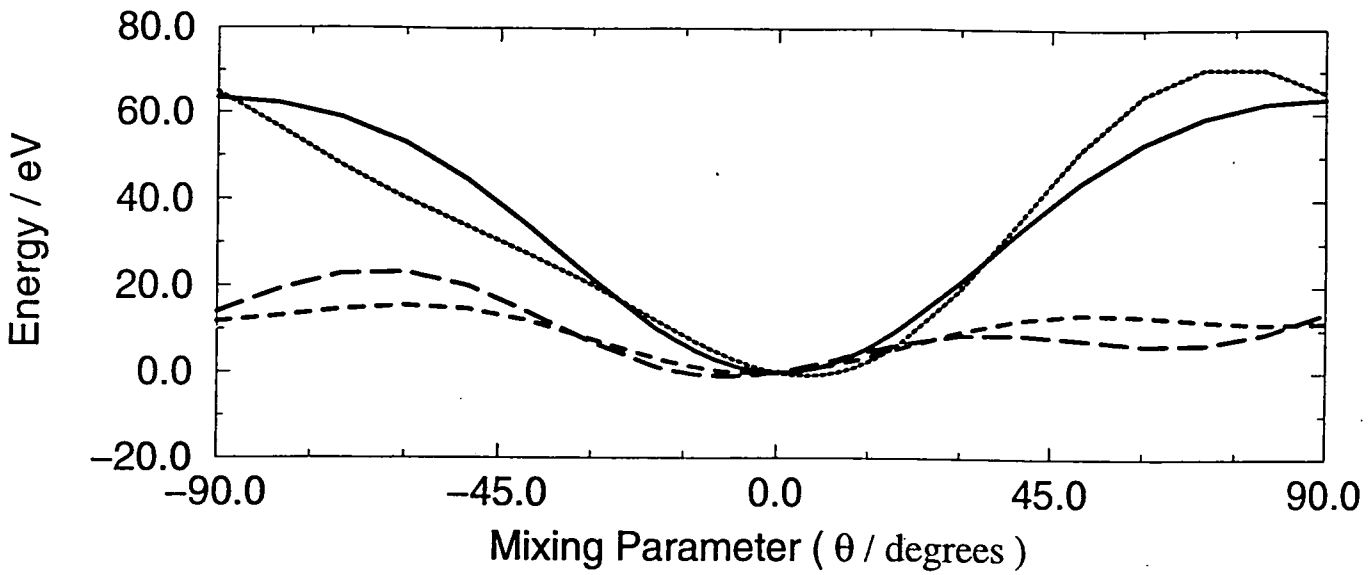
Fock matrix (orbital energies)

-35.83281	-10.32503	-10.10889	-10.10889	.03437	2.40056	2.40056	5.80529
-----------	-----------	-----------	-----------	--------	---------	---------	---------

the HOMO is doubly degenerate (a π orbital) and cannot be used in our 2-orbital approximation. There are four possible combinations of occupied and unoccupied orbitals that we can use. These are orbitals 1 & 5, orbitals 1 & 8, orbitals 2 & 5 and finally orbitals 2 & 8. For each combination of AOMO and AUMO, the molecular energy of the solid state was calculated using the **Solid** program for values of θ between $\theta = -90^\circ$ and $\theta = +90^\circ$, at 10° intervals. This was then added to the electrostatic energy from the **Ewald** program to give the total energy. The upper panel of Figure 1 shows the total energy for the four combinations of orbitals between $\theta = -90^\circ$ and $\theta = +90^\circ$. As the region around $\theta = 0$ is of particular interest, further calculations of the molecular, electrostatic and total energy were made between $\theta = -10^\circ$ and $\theta = +10^\circ$, at intervals of $\theta = 0.5^\circ$. The lower panel of Figure 1 shows an expanded view of the total energies in the region between $\theta = -10^\circ$ and $\theta = +10^\circ$.

Total Energy

for orbital combinations



Total Energy

for orbital combinations

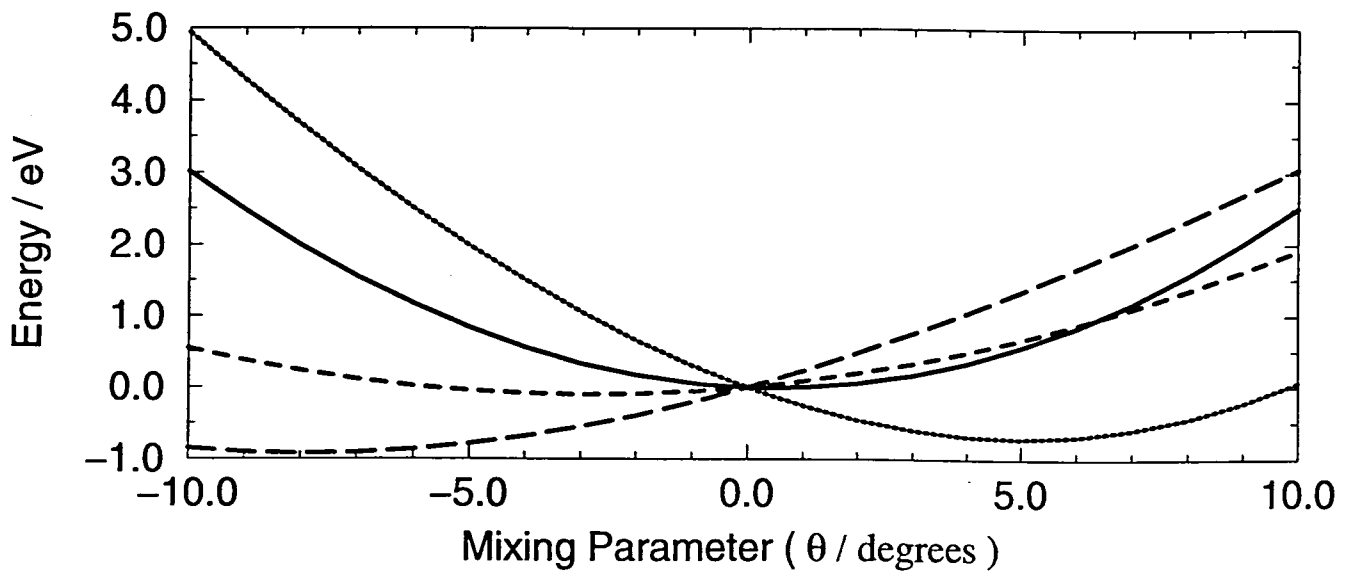
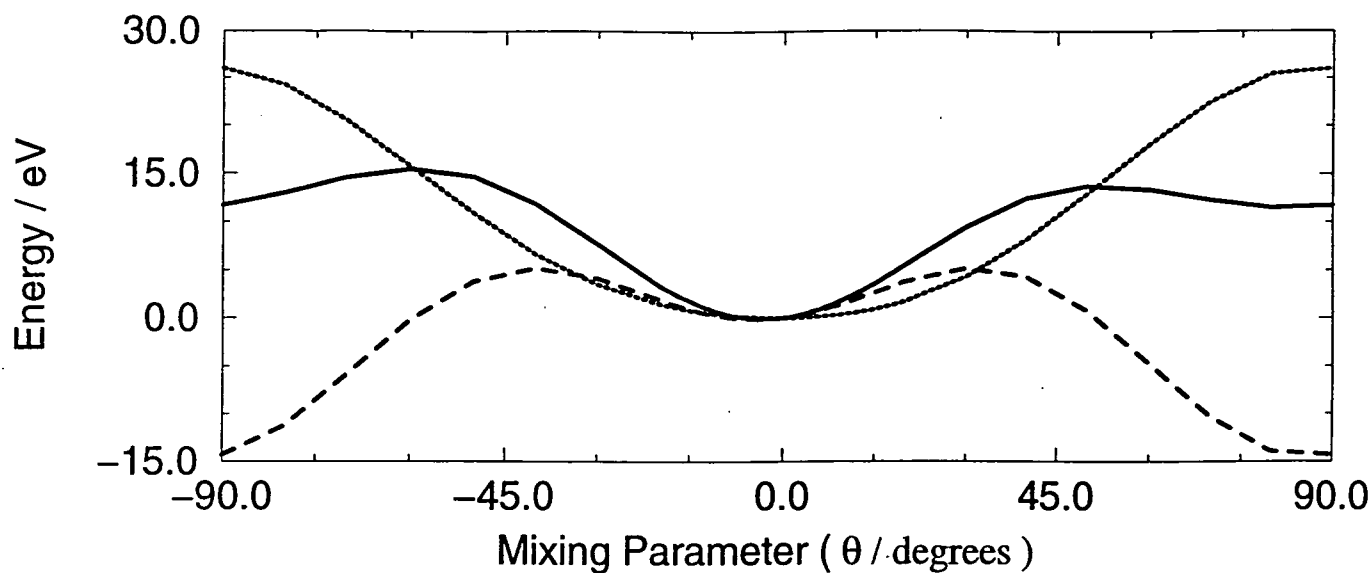


Figure 1. The total solid-state energy as a function of the mixing parameter θ for the possible combinations of active orbitals, orbitals 1 & 5, orbitals 1 & 8, orbitals 2 & 5 and orbitals 2 & 8.

Energy Components

Orbitals 2 & 5



..... Molecular
 ----- Electrostatic
 ——— Total

Energy Components

Orbitals 2 & 5

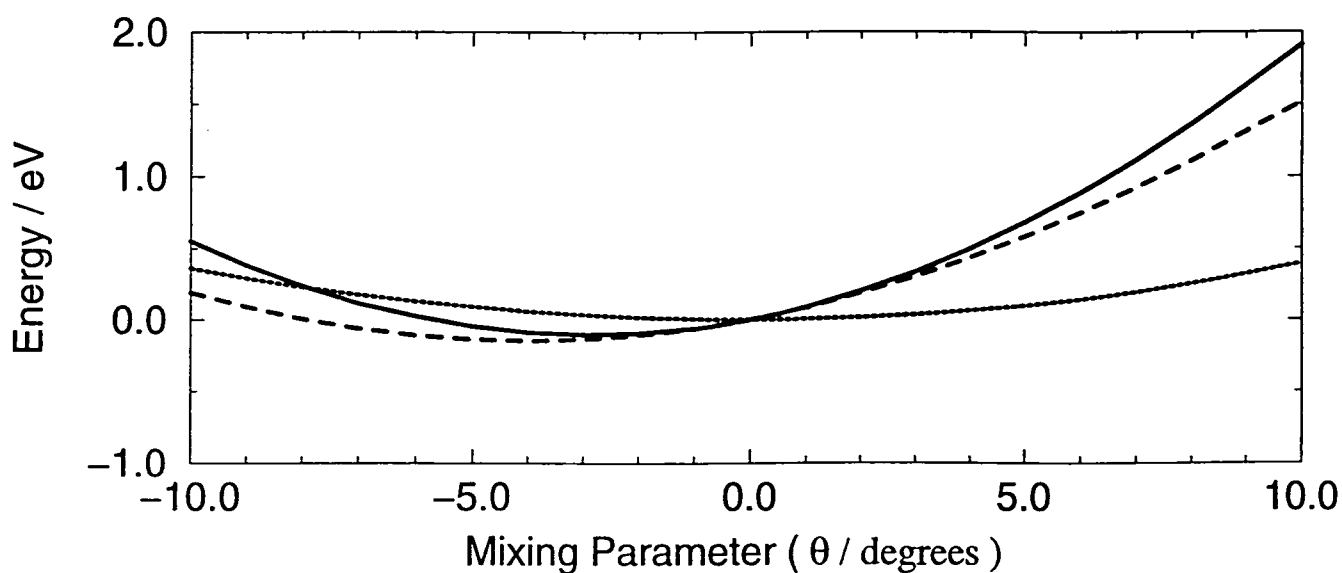
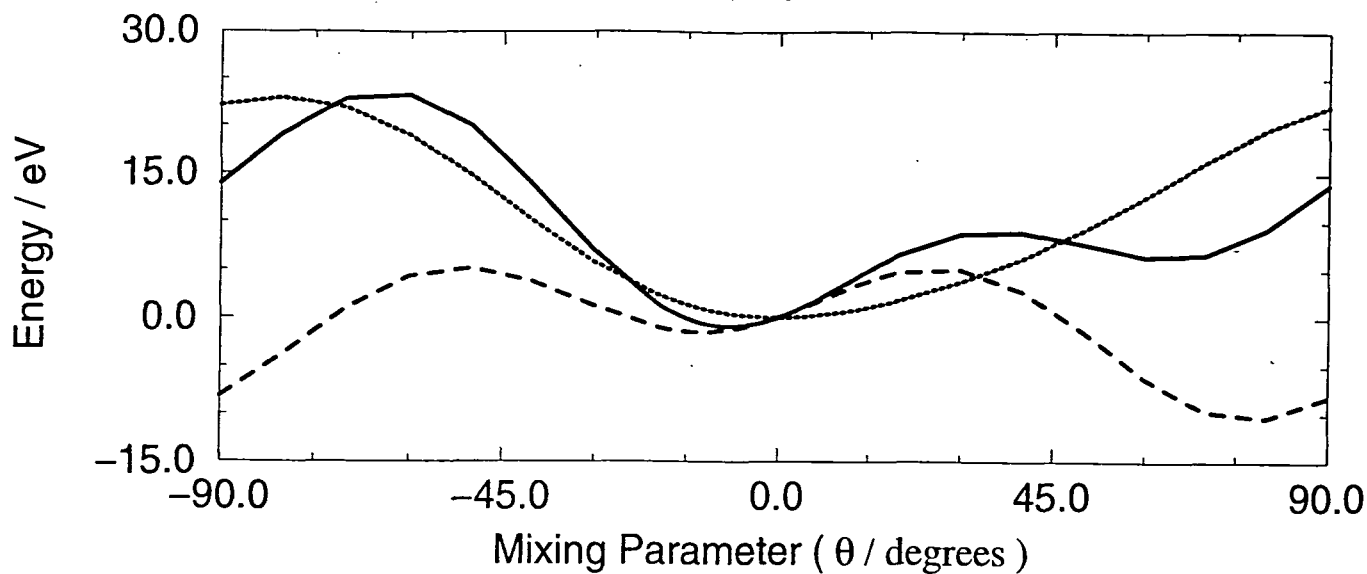


Figure 2. The molecular, electrostatic and total solid-state energy as a function of the mixing parameter θ for the active orbitals 2 & 5.

10: Results from the Solid Program

Energy Components

Orbitals 2 & 8



— Molecular
 --- Electrostatic
 ... Total

Energy Components

Orbitals 2 & 8

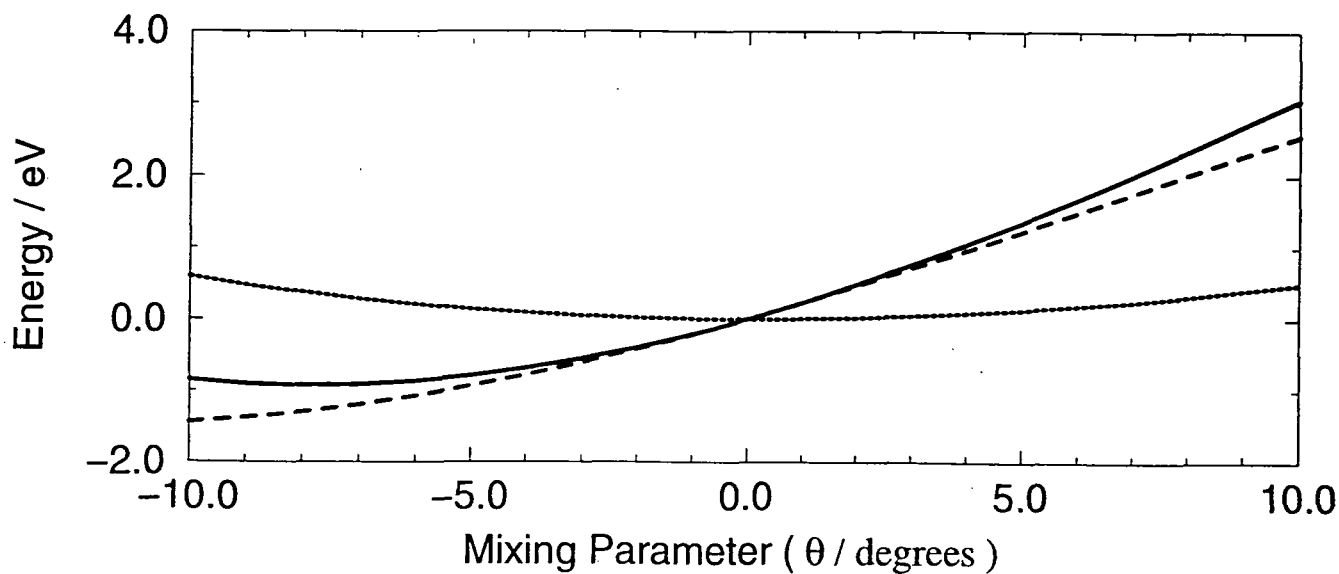


Figure 3. The molecular, electrostatic and total solid-state energy as a function of the mixing parameter θ for the active orbitals 2 & 8.

10: Results from the Solid Program

The total energies and charges at the lowest energy point for each combination of orbitals mixing are shown in Table 6.

Table 6 Energies and charges for various orbital combinations

Orbitals	Theta / degrees	Energy / eV	Charge
1 & 5	0.5	-0.0067	0.6448
1 & 8	5.0	-0.7162	0.7178
2 & 5	-3.5	-0.1064	0.6526
2 & 8	-8.0	-0.9238	0.7202

It may be seen that mixing orbitals 1 & 8 or 2 & 8 gives the greatest stabilisation, with a significant increase in ionicity in each case. The energies shown are referred to the energy at $\theta = 0^\circ$, when there is no mixing of orbitals. The charges given are the charges of the Lithium for each combination of orbitals. Since the system is diatomic the charges on the atoms are equal and opposite. The charge for the gas phase, i.e. $\theta = 0^\circ$, is 0.6441. LiF is considered to be perfectly ionic in the solid-state phase and therefore the solid-state charge usually should be 1.0. Our model does not fully reproduce this transfer of charge in the solid state. This is because there is not enough flexibility in our model, in which the charge transfer is modelled by only two active orbitals, one occupied and one unoccupied.

The molecular energy of the mixed system will always be larger than that of the gas phase for any non-zero value of θ , and therefore any overall lowering of energy must be caused by the electrostatic energy. For the orbital mixing combinations 2 & 5 and 2 & 8, Figures 2 and 3 show the molecular, electrostatic and total energies. In each figure the upper panel is between $\theta = -90^\circ$ and $\theta = +90^\circ$ and the lower panel highlights the region between $\theta = -10^\circ$ and $\theta = +10^\circ$.

The electrostatic energy for each combination of orbitals has two minima, one of which is close to $\theta = 0$, and both of which correspond to a negative electrostatic energy. The minimum close to $\theta = 0$ always occurs when the Lithium charge is more positive than in the gas phase. The other minimum occurs when the Lithium charge is very negative and can sometimes actually give a more negative energy than the minimum close to $\theta = 0$.

10: Results from the Solid Program

The molecular energy always has to be positive, increasing as one moves away from $\theta = 0$. Therefore, the total energy, as the sum of the two, will only be negative at the minimum that is close to $\theta = 0$.

PART VI

Future Developments

11: Future Development

Currently, the model that we have developed can only handle mixing of two orbitals. It is known that this is not enough to reproduce some physical effects.⁵⁵ A more complete model could be implemented in two ways. The first, and simplest, would be to mix the orbitals again once the first mixing had been performed. This would be consistent with the mathematics that we have developed, but would require an unattractive iteration procedure. The second would involve mixing several orbitals together in one step. This would mean extending the mathematics of the model quite considerably.

The model could use an *ab initio* package to perform the initial calculations, rather than a semi-empirical one as at present. There is no restriction in the mathematics to either one or the other; any Hartree-Fock method can be used. The method of calculating charges as implemented within our program would have to be changed for an *ab initio* method, as there is no neglect of differential overlap, but that should not pose a problem. Initial studies have shown that some of the problems we have encountered with combining semi-empirical orbitals would still be present in an *ab initio* implementation, but the overall calculations would be more accurate. A method of calculating charges other than that of Mulliken could also be used.

At present the model assumes that the geometry of the molecule does not change as the system goes from the gas phase to the solid phase. Allowing the geometry to change, and indeed optimising the geometry, would allow more complete investigation of the effects of the change in environment.

The programs could be rewritten to ensure they are integrated more fully than they currently are. This would involve calculating the Ewald energy within the **Solid** program rather than the current cumbersome method of writing out a file and then executing the **Ewald** program. Along the same lines the **Solid** program should be able to produce results for a range of θ values, rather than a single value as at present. A more fundamental development would be for the **Solid** program to minimise the total energy, i.e. the sum of the molecular and Ewald energies, and thus to determine an optimum degree of mixing directly.

Appendix A: Atomic Units and Fundamental Constants

In quantum chemistry it is normal to work with fundamental properties on the atomic scale. These value are reported in Table 7.

Table 7 Atomic units⁵⁶

Atomic Units		
e	Elementary charge	$1.60217733 \times 10^{-19} \text{ C}$
a_0	Bohr radius	$5.29177249 \times 10^{-11} \text{ m}$
E_h	Hartree energy	$4.3597482 \times 10^{-18} \text{ J}$

Also several fundamental constants are used in this thesis. Table 8 shows the values of these constants.

Table 8 Fundamental Constants⁵⁶

Fundamental Constants		
ϵ_0	Permittivity of vacuum	$8.854187816 \times 10^{-12} \text{ Fm}^{-1}$
N_A	Avogadro number	$6.0221367 \times 10^{23} \text{ mol}^{-1}$

Appendix B: Program Listing of Ewald

```

*
* Main program
*
      program ewald
      integer maxat,index,numat,icells,molcul,numion,index,i
      parameter (maxat=30)
      integer iontyp(maxat)
      double precision axes(3,3),angles(3),atoms(4,maxat),totmad,volume
      double precision rmadlg,rspac,accrcy,espace,emadlg,rmadt1,elctrn
      double precision gvec(3,3),emadt1,factor,eta
      double precision length(3),atomfr(4,maxat),epsln0,pi
      double precision rmin,bohr,hartre,charge(2),oldrmg,oldemg
      real dtime,tarray(2),idummy
      logical fractn
      character*80 string,title,label(2)

*
* Set up constants
*
      pi = acos(-1.0d0)
      epsln0 = 8.854187816d-12
      elctrn = 1.60217733d-19
      bohr   = 5.29177249d-11
      hartre = 4.3597482d-18

*
* call routine that reads in the data
*
      eta = -1
      call reader(length,angles,atomfr,numat,eta,accrcy,numion
+ ,fractn,title,iontyp,label,charge,molcul)
      call recip (length,angles,gvec,axes,volume)

*
* Write out angstrom coordinates
*
      if (fractn) then
        do index = 1,numat
          call frtoan(atoms(1,index),atomfr(1,index),axes)
          atoms(4,index) = atomfr(4,index)
        enddo
        call block(0,1,'Bohr atom data')
        write(string,9990) 'Atom data','x','y','z','charge'
        call block(0,3,string)
        do 21 index = 1,numat
          write (string,9980) 'Atom',index,(atoms(i,index),i=1,4)
          call send(0,string)
        enddo
      end if

```

```

21  continue
    endif
*
9990 format (a9,9x,a1,13x,a1,13x,a1,11x,a6)
9980 format (a4,i4,4f14.5)
*
* Set to best value of eta if not read in
*
    if (eta.eq.-1) then
        eta = dble( ( numat * pi**3 ) / volume ** 2 ) ** (1.0d0/3.0d0)
        write (string,*) 'The best value of eta is',eta
    else
        write (string,*) 'You have requested a value of eta of ',eta
    endif
    call block(1,1,string)
*
* Accuracy
*
    write (string,*) 'The accuracy is set at',accrcy
    call block(1,1,string)
*
    factor = 0.0d0
    do 50 index =1,numat
        factor = factor + ( atoms(4,index)**2 * sqrt(eta/pi) )
50  continue
*
* use function that calculates the inverse space component
*
    call block(1,1,'Reciprocal Space component')
    icells = 0
    emadt1 = -factor
20  icells = icells + 1
    idummy= dtime(tarray)
    emadlg = espace(axes,atoms,numat,icells,eta,gvec,volume)
    oldemg = emadt1
    emadt1 = emadt1 + emadlg
    write (string,*) 'The value for icells is',icells
    call send(0,string)
    call report(0,emadt1,molcul)
    if (dabs((emadt1-oldemg)/emadt1).gt.accrcy) goto 20
    idummy = dtime(tarray)
    emadt1 = emadt1
    write (string,*) 'Convergence was achieved, icells = ',icells
    call send(1,string)
    call report(1,emadt1,molcul)
    write (string,'(a10,f10.6,a8)') 'This took ',tarray(1),' seconds'
    call send(1,string)
*

```

Appendix B: Program Listing of Ewald

```

* use function that calculates the real space component
*
      rmin = 100.0d0
      call block(1,1,'Real Space component')
      icells = -1
      rmadtl = 0.0d0
30   icells = icells + 1
      idummy = dtime(tarray)
      rmadlg = rspace(axes,angles,atomfr,numat,icells,eta,rmin)
      oldrmg = rmadtl
      rmadtl = rmadlg
      write (string,*) 'The value for icells is',icells
      call send(0,string)
      call report(0,rmadtl,molcul)
      if (dabs((rmadtl-oldrmg)/rmadtl).gt.accrcy) goto 30
      idummy= dtime(tarray)
      write (string,*) 'Convergence was achieved, icells = ',icells
      call send(1,string)
      call report(1,rmadtl,molcul)
      write (string,'(a10,f10.6,a8)') 'This took ',tarray(1),' seconds'
      call send(1,string)
*
* add the two together
*
      totmad = rmadtl + emadtl
      call block(1,1,'Total energy')
      call report(1,totmad,molcul)
      call block(1,1,'Madelung constant')
      rmin = rmin * bohr
      write (string,*) 'Smallest ion-ion distance',rmin*1.0d10
      call send(1,string)
      write (string,*) 'The Madelung Constant is',
+   totmad * hartre * 4.0d0 * pi * epsln0 * rmin / elctrn**2
+   / charge(1) / charge(2) / molcul
      call send(1,string)
end
*
*
*
      subroutine block(screen,type,line)
      character*(*) line
      integer i,type,len,lnblnk,spaces,width
      logical screen
      character*80 string
      parameter (width=65)
      if (type.eq.1) then
        len = lnblnk(line)
        write (string,*) ('*',i=1,width)

```

Appendix B: Program Listing of Ewald

```

        call send(screen,string)
        write(string,*)' ' ,line(1:len),(' ',i=1,width-3-len),'*'
        call send(screen,string)
        write (string,*) ('*',i=1,width)
        call send(screen,string)
    endif
    if (type.eq.2) then
        len = lnblnk(line)
        spaces = (width - 3 - 2*len) / 2
        write (string,*) ('*',i=1,width)
        call send(screen,string)
        write (string,*) '**',(' ',i=1,width-2),'*'
        call send(screen,string)
        write(string,*)' ' ,(' ',i=1,spaces),(line(i:i),' ',i=1,len)
+      ,(' ',i=1,spaces),'*'
        call send(screen,string)
        write (string,*) '**',(' ',i=1,width-2),'*'
        call send(screen,string)
        write (string,*) ('*',i=1,width)
        call send(screen,string)
    endif
    if (type.eq.3) then
        call send(screen,line)
        write (string,*) (' ',i=1,65)
        call send(screen,string)
    endif
    return
end

*
*
*
subroutine cross(vec,a,b)
double precision vec(3),a(3),b(3)
vec(1) = a(2)*b(3) - a(3)*b(2)
vec(2) = a(3)*b(1) - a(1)*b(3)
vec(3) = a(1)*b(2) - a(2)*b(1)
return
end

*
*
*
double precision  function erfc ( x )

*
* approximation to the complementary error function
* reference:
*   abramowitz and stegun, handbook of mathematical functions,
*   national bureau of standards, formula 7.1.26
*

```

```

double precision a1, a2, a3, a4, a5, p,t, x, xsq, tp,erfc
parameter ( a1 = 0.254829592d0, a2 = -0.284496736d0 )
parameter ( a3 = 1.421413741d0, a4 = -1.453152027d0 )
parameter ( a5 = 1.061405429d0, p = 0.3275911d0 )
t = 1.0 / ( 1.0 + p * x )
xsq = x * x
tp = t * ( a1 + t * ( a2 + t * ( a3 + t * ( a4 + t * a5 ) ) ) )
erfc = tp * exp ( -xsq )
return
end

*
*
*

double precision
+ function espace(axes,atoms,numat,icells,eta,gvec,volume)
double precision axes(3,3),atoms(4,*),eta,espace,expon
double precision gvec(3,3),gx,gy,gz,gsq,gdotrt,rtx,rty,rtz
double precision rix,riy,riz,pi,volume,gdotri
double complex sum,energy,potent,varab
integer numat,icells,index,jindex,kndex,loopi,loopt
pi = acos(-1.0d0)
energy = ( 0.0d0 , 0.0d0 )
do 10 loopi = 1,numat
  potent = ( 0.0d0 , 0.0d0 )
  rix = atoms(1,loopi)
  riy = atoms(2,loopi)
  riz = atoms(3,loopi)
  do 20 index = -icells,icells
    do 20 jindex = -icells,icells
      do 20 kndex = -icells,icells
        gx = index*gvec(1,1) + jindex*gvec(1,2) + kndex*gvec(1,3)
        gy = index*gvec(2,1) + jindex*gvec(2,2) + kndex*gvec(2,3)
        gz = index*gvec(3,1) + jindex*gvec(3,2) + kndex*gvec(3,3)
        if ((iabs(index).eq.icells).or.(iabs(jindex).eq.icells)
+       .or.(iabs(kndex).eq.icells)) then
          sum = ( 0.0d0,0.0d0 )
          do 30 loopt = 1,numat
            rtx = atoms(1,loopt)
            rty = atoms(2,loopt)
            rtz = atoms(3,loopt)
            gdotrt = gx*rtx + gy*rty + gz*rtz
            sum=sum+atoms(4,loopt)*cmplx(cos(gdotrt),-sin(gdotrt))
30          continue
          gdotri = gx*rix + gy*riy + gz*riz
          varab = cmplx(cos(gdotri),sin(gdotri))
          gsq = gx*gx + gy*gy + gz*gz
          expon = exp(-0.25d0*gsq/eta)/gsq
          potent = potent+sum*expon*varab

```

Appendix B: Program Listing of Ewald


```

endif
20 continue
    energy = energy + potent * atoms(4,loopi)
10 continue
    espace = energy * 4.0d0 * pi / volume
    espace = espace / 2.0d0
    return
end

*
*
*

subroutine frtoan (pos,frc,axes)
double precision pos(3),axes(3,3),frc(3)
pos(1) = frc(1)*axes(1,1) + frc(2)*axes(1,2) + frc(3)*axes(1,3)
pos(2) = frc(1)*axes(2,1) + frc(2)*axes(2,2) + frc(3)*axes(2,3)
pos(3) = frc(1)*axes(3,1) + frc(2)*axes(3,2) + frc(3)*axes(3,3)
return
end

*
*
*

subroutine reader(length,angles,atomfr,numat,eta,accrcy
+      ,numion,fractn,title,iontyp,label,charge,molcul)
double precision length(3),angles(3),atomfr(4,*),a0,eta,pi,accrcy
double precision charge(2)
integer index,i,numat,numion,iontyp(*),molcul
character*80 string,keywrd,qualif,title,label(*)
logical fractn
a0 = 5.29177249d-11
pi = acos(-1.0d0)
fractn = .false.
accrcy = 1.0d-10

*
* loop around this until end
*
    10 read (5,'(a80)') string
        read (string,*) keywrd

*
* read molecule number
*
        if (string(1:4).eq.'mole') read (string,*) keywrd,molcul

*
* Read ion data
*
        if (string(1:4).eq.'ions') then
            read (string,*) keywrd,numion
            do 11 index = 1,numion
                read (5,*) i,label(index),charge(index)

```

```

11 continue
endif
*
* Read length data
*
  if (string(1:4).eq.'leng') then
    read (string,*) keywrd,qualif,(length(i),i=1,3)
    if (qualif(1:4).eq.'angs') then
      do 30 index = 1,3
        length(index) = length(index) / ( a0 * 1.0d10 )
30    continue
      else
        stop 'Error in reading length data'
      endif
    endif
  endif
*
* Read angles data
*
  if (string(1:4).eq.'angl') then
    read (string,*) keywrd,qualif,(angles(i),i=1,3)
    if (qualif(1:4).eq.'degr') then
      do 50 index = 1,3
        angles(index) = pi * angles(index) / 180.0d0
50    continue
      else
        stop 'Error in reading angle data'
      endif
    endif
  endif
*
* Read atom data
*
  if (string(1:4).eq.'atom') then
    read (string,*) keywrd,qualif,numat
    do 20 index = 1,numat
      read (5,*) (atomfr(i,index),i=1,3),iontyp(index)
      atomfr(4,index) = charge(iontyp(index))
20    continue
    if (qualif(1:4).eq.'frac') then
      fractn = .true.
    else
      stop 'Error in reading atom data'
    endif
  endif
*
* Read in value of eta
*
  if (string(1:4).eq.'eta ') read (string,*) keywrd,eta
*

```

Appendix B: Program Listing of Ewald

```

* Read in title
*
      if (string(1:4).eq.'titl') title = string(7:80)
*
* Set the accuracy
*
      if (string(1:4).eq.'accu') read (string,*) keywrd,accrcy
*
* Hit the 'end' therefore exit
*
      if (string(1:3).ne.'end') goto 10
*
* test to see data is read in correctly
*
      call block(1,2,title)
      call block(1,1,'Input data')
      write(string,9990) 'Axes   (in bohr)      ',(length(i),i=1,3)
      call block(1,3,string)
      write(string,9990) 'Angles (in Radians)  ',(angles(i),i=1,3)
      call block(1,3,string)
      write(string,9980) 'Atom data','x','y','z','charge'
      call block(1,3,string)
      do 40 index = 1,numat
        write (string,9970) 'Atom',index,(atomfr(i,index),i=1,4)
        call send(1,string)
      40 continue
9990 format (a22,3f14.5)
9980 format (a9,9x,a1,13x,a1,13x,a1,11x,a6)
9970 format (a4,i4,4f14.5)
      return
      end
*
*
*
      subroutine recip(length,angles,gvec,axes,volume)
      integer index,jindex
      double precision gvec(3,3),axes(3,3),twopi,volume,angles(3)
      double precision length(3),dummy(3)
      character*80 string
*
* Calculate A vector
*
      axes(1,1) = length(1)
      axes(2,1) = 0.0d0
      axes(3,1) = 0.0d0
*
* Calculate B vector
*

```

Appendix B: Program Listing of Ewald

```

axes(1,2) = length(2) * cos(angles(3))
axes(2,2) = length(2) * sin(angles(3))
axes(3,2) = 0.0d0
*
* Calculate C vector
*
axes(1,3) = length(1) * length(3) * cos(angles(2)) / axes(1,1)
axes(2,3) = ( ( length(2) * length(3) * cos(angles(1)) )
+           - ( axes(1,2) * axes(1,3) ) ) / axes(2,2)
axes(3,3) = sqrt ( length(3)**2 - axes(1,3)**2 - axes(2,3)**2 )
*
* Make sure they are all zero
*
do 9 index = 1,3
  do 9 jindex = 1,3
    if(abs(axes(index,jindex)).lt.5.0d-15) axes(index,jindex)=0.0d0
  9 continue
*
* Calculate the reciprocal vector
*
call cross(gvec(1,1),axes(1,2),axes(1,3))
call cross(gvec(1,2),axes(1,3),axes(1,1))
call cross(gvec(1,3),axes(1,1),axes(1,2))
call cross(dummy(1) ,axes(1,2),axes(1,3))
volume =axes(1,1)*dummy(1)+axes(2,1)*dummy(2)+axes(3,1)*dummy(3)
twopi = 2.0d0 * acos(-1.0d0)
do 10 index = 1,3
  do 10 jindex = 1,3
    gvec(index,jindex) = gvec(index,jindex) * twopi / volume
  10 continue
call block(0,1,'Axes ( Bohr)')
do 20 index = 1,3
  write (string,'(3f10.6)') (axes(jindex,index),jindex=1,3)
  call send (0,string)
20 continue
call block(0,1,'Gvec (Reciprocal Bohr)')
do 30 index = 1,3
  write (string,'(3f10.6)') (gvec(jindex,index),jindex=1,3)
  call send (0,string)
30 continue
return
end
*
*
*
subroutine report(screen,energy,molcul)
double precision energy,hartre,avdro
integer molcul

```

```

logical screen
character*80 string
hartre = 4.3597482d-18
avgd20 = 6.0221367d20
write(string,990) 'The energy in Hartrees is',energy/molcul
call send(screen,string)
write(string,980)'The energy in Joules is      ',energy*hartre/molcul
call send(screen,string)
write(string,990)
+   'The energy in kJ/mol is      ',energy*hartre*avgd20/molcul
call send(screen,string)
990 format (a25,f17.8)
980 format (a25,d21.8)
return
end

*
*
*
double precision function
+   rspace(axes,angles,atomfr,numat,icells,eta,rmin)
double precision axes(3,3),angles(3),atomfr(4,*),eta,rspace,c,i
double precision dist,erfc,xfr,yfr,zfr,rmin,value,vec(3),vecfr(3)

*
double precision a0

*
integer numat,icells,loop,loop2,index,jindex,kndex
rspace = 0.0d0
rewind (98)
rewind (99)
a0 = 0.529177249
do 10 loop = 1,numat
  c = atomfr(4,loop)
  do 10 index = -icells,icells
    do 10 jindex = -icells,icells
      do 10 kndex = -icells,icells
        xfr = atomfr(1,loop)+index
        yfr = atomfr(2,loop)+jindex
        zfr = atomfr(3,loop)+kndex
*       if ((iabs(index).eq.icells).or.(iabs(jindex).eq.icells)
*       +   .or.(iabs(kndex).eq.icells)) then
        if (.true.) then
          do 20 loop2 = 1,loop
            vecfr(1) = atomfr(1,loop2) - xfr
            vecfr(2) = atomfr(2,loop2) - yfr
            vecfr(3) = atomfr(3,loop2) - zfr
            call frtoan(vec,vecfr,axes)
            dist = sqrt(vec(1)**2 + vec(2)**2 + vec(3)**2)
            if (dist.gt.1.0d-2) then

```

Appendix B: Program Listing of Ewald

```

*   write (99,9990) dist*a0,dist,index,jindex,kndex,loop,loop2
*   write (99,9980) xfr,yfr,zfr,(atomfr(i,loop2),i=1,3)
*   write (99,9980) (vecfr(i),i=1,3),(vec(i)*a0,i=1,3)
*   write (99,*)
*   write (98,9990) dist*a0,dist,index,jindex,kndex,loop,loop2
      if (dist.lt.rmin) rmin = dist
      value = atomfr(4,loop2)*c*erfc(sqrt(eta)*dist)/dist
      rspace = rspace + value
      if (loop.eq.loop2) rspace = rspace - value /2.0d0
    endif
20      continue
    endif
10 continue
9990 format (2f12.8,5i5)
9980 format (3f12.8,7x,3f12.8)
    return
    end
*
*
*
    subroutine send(screen,string)
    logical screen
    character*(*) string
    if (screen) then
      write (6,'(a65)') string(1:65)
      write (12,'(a75)') string(1:65)
    endif
    write (13,'(a75)') string(1:65)
    call flush(12)
    call flush(13)
    return
    end

```

Appendix C: Ewald Program Data Set for NaCl

```
title Unit Cell for NaCl
length angstroms 5.64056 5.64056 5.64056
angles degrees 90.0 90.0 90.0
ions 2
    1      Na      +1
    2      Cl      -1
atoms fractional 8
    0.0    0.0    0.0    1
    0.5    0.5    0.0    1
    0.0    0.5    0.5    1
    0.5    0.0    0.5    1
    0.5    0.5    0.5    2
    0.5    0.0    0.0    2
    0.0    0.5    0.0    2
    0.0    0.0    0.5    2
accuracy 1.0d-5
molecule 4
end
```

Appendix D: Ewald Program Output for NaCl

```

*****
*
*           U n i t   C e l l   f o r   N a C l
*
*****
* Input data
*****
Axes   (in bohr)           10.65911      10.65911      10.65911

Angles (in Radians)        1.57080      1.57080      1.57080

Atom data      x           y           z           charge

Atom  1      ) 0.00000      0.00000      0.00000      1.00000
Atom  2      0.50000      0.50000      0.00000      1.00000
Atom  3      0.00000      0.50000      0.50000      1.00000
Atom  4      0.50000      0.00000      0.50000      1.00000
Atom  5      0.50000      0.50000      0.50000      -1.00000
Atom  6      0.50000      0.00000      0.00000      -1.00000
Atom  7      0.00000      0.50000      0.00000      -1.00000
Atom  8      0.00000      0.00000      0.50000      -1.00000

*****
* The best value of eta is  5.5301606830913D-02
*****
* The accuracy is set at  1.0000000000000D-05
*****
* Reciprocal Space component
*****
Convergence was achieved, icells =  2
The energy in Hartrees is      -0.25962997
The energy in Joules is        -0.11319213D-17
The energy in kJ/mol is        -681.65846416
This took  2.840000 seconds

*****
* Real Space component
*****
Convergence was achieved, icells =  2
The energy in Hartrees is      -0.06827058
The energy in Joules is        -0.29764256D-18
The energy in kJ/mol is        -179.24441765
This took  1.160000 seconds

*****

```

Appendix D: Ewald Program Output for NaCl


```
* Total energy *
*****
The energy in Hartrees is      -0.32790055
The energy in Joules is        -0.14295638D-17
The energy in kJ/mol is        -860.90288181
*****
* Madelung constant *
*****
Smallest ion-ion distance      2.8202800000000
The Madelung Constant is       1.7475644799243
```

Appendix E: Mopac Subroutine solidorb.f

```

subroutine solidorb
implicit double precision (a-h,o-z)
include 'sizes'
common /vector/ c(morb2),eigs(maxorb),cbeta(morb2),eigb(maxorb)
common /molkst/ numat,nat(numatm),nfirst(numatm),nmidle(numatm),
1          nlast(numatm), norbs, nelecs,nalpha,nbeta,
2          nclose,nopen,ndumy,fract
common /wmatrx/ wj(n2elec), wk(n2elec)
common /keywrd/ keywrd
common /elems/ elemnt(107)
common /coord / coord(3,numatm)
common /hmatrx/ h(mpack)
dimension w(n2elec),eiga(maxorb),occa(maxorb)
dimension rjay(maxorb,maxorb),rkay(maxorb,maxorb)
dimension orbs(maxorb,maxorb),triple(maxorb)
dimension aoone(maxorb,maxorb)
double precision moone(maxorb,maxorb)
equivalence (w,wj)
real wj, wk
character*80 keywrd
character*2 elemnt(107)

*
* First of all, make a matrix of orbitals from c
*
      do i = 1,norbs
        do j = 1,norbs
          orbs(i,j) = c(i+((j-1)*norbs))
        enddo
      enddo

*
* Then, calculate the J & K integrals
*
      do i = 1,norbs
        do j = 1,norbs
          rjay(i,j) = spcg(orbs(1,i),orbs(1,i),orbs(1,j),orbs(1,j),w,wj)
          rkay(i,j) = spcg(orbs(1,i),orbs(1,j),orbs(1,j),orbs(1,i),w,wj)
        enddo
      enddo

*
* Now we need all of the 1 electron integrals
* First of all the ao
*
      icount = 0
      do i = 1,norbs

```

Appendix E: Mopac Subroutine solidorb.f

```

        do j = 1,i
            icount = icount + 1
            aoone(i,j) = h(icount)
            aoone(j,i) = h(icount)
        enddo
    enddo

*
* Now transform to the mo basis
*

    do i = 1,norbs
        do j = 1,norbs
            moone(i,j) = 0.0d0
            do k = 1,norbs
                do l = 1,norbs
                    moone(i,j)=moone(i,j) + orbs(k,i) * aoone(k,l) * orbs(l,j)
                enddo
            enddo
        enddo
    enddo

*

    open (unit=50,file='for050',status='unknown')
    open (unit=51,file='for051',status='unknown',form='unformatted')

*

    write (50,*) 'RHF Orbitals',.true.
    write (50,*) 'Number of atoms is ',numat
    write (50,*) 'Number of orbitals is ',norbs
    write (50,*) 'Number of closed orbitals is',nclose
    write (50,*) 'Number of open orbitals is',nopen - nclose
    write (50,*) 'Number of orbitals in CI is',nmos
    write (50,*) 'Active Occupied Molecular Orbital is',nclose
    write (50,*) 'Active Unoccupied Molecular Orbital is',nclose+1

*

    write (51) .true.
    write (51) numat
    write (51) norbs
    write (51) nclose
    write (51) nopen - nclose
    write (51) nmos
    write (51) nclose
    write (51) nclose+1

*

    write (50,*) 'For each atom, the first and last orbital'
    do i = 1,numat
        write (50,9990) i,nfirst(i),nlast(i),nat(i),elemnt(nat(i))
        write (51)      i,nfirst(i),nlast(i),nat(i),elemnt(nat(i))
    enddo

*

    write (50,*) 'Orbital co-efficients (column vectors)'

```

Appendix E: Mopac Subroutine solidorb.f

```

do i = 1,norbs
  write (50,9980) (orbs(i,j),j=1,norbs)
  write (51)      (orbs(i,j),j=1,norbs)
enddo

*
write(50,*)      'Fock matrix (orbital energies)'
write(50,9980) (eigs(i),i=1,norbs)
write(51)      (eigs(i),i=1,norbs)

*
*
write(50,*) 'one-electron matrix - mo basis'
do i=1,norbs
  write(50,9970) i,(moone(i,j),j=1,norbs)
  write(51)      i,(moone(i,j),j=1,norbs)
enddo

*
write(50,*) 'two-electron j-integrals'
do i=1,norbs
  write(50,9970) i,(rjay(i,j),j=1,norbs)
  write(51)      i,(rjay(i,j),j=1,norbs)
enddo

*
write(50,*) 'two-electron k-integrals'
do i=1,norbs
  write(50,9970) i,(rkay(i,j),j=1,norbs)
  write(51)      i,(rkay(i,j),j=1,norbs)
enddo

*
* Now the triple ( su | lkk l).
* I calculate a line and then write it out.
*
write(50,*) 'triple ( su | kk )'
do i = 1,norbs
  do j = 1,norbs
    do k = 1,norbs
      triple(k) = spcg(orbs(1,i),orbs(1,j),orbs(1,k),orbs(1,k),w,wj)
    enddo
    write(50,9960) i,j,(triple(k),k=1,norbs)
    write(51)      i,j,(triple(k),k=1,norbs)
  enddo
  write (50,*)
enddo

*
close (50)
close (51)

*
9990 format (4i5,5x,a5)
9980 format (100f11.5)

```

```
9970 format (i5,100f11.5)
9960 format (2i5,100f11.5)
*
    end
```

Appendix F: .orb output file for LiF

RHF Orbitals T

Number of atoms is 2

Number of orbitals is 8

Number of closed orbitals is 4

Number of open orbitals is 0

Number of orbitals in CI is 2

Active Occupied Molecular Orbital is 2

Active Unoccupied Molecular Orbital is 5

For each atom, the first and last orbital

1	1	4	3	li
2	5	8	9	f

Orbital co-efficients (column vectors)

0.14884	-0.15960	0.00000	0.00000	-0.87303	0.00000	0.00000	-0.43611
0.21924	-0.15254	0.00000	0.00000	0.48209	0.00000	0.00000	-0.83442
0.00000	0.00000	-0.17173	0.00355	0.00000	-0.98493	-0.02035	0.00000
0.00000	0.00000	0.00355	0.17173	0.00000	0.02035	-0.98493	0.00000
0.96312	0.10632	0.00000	0.00000	0.02177	0.00000	0.00000	0.24620
-0.04662	0.96951	0.00000	0.00000	-0.07025	0.00000	0.00000	-0.23008
0.00000	0.00000	-0.98493	0.02035	0.00000	0.17173	0.00355	0.00000
0.00000	0.00000	0.02035	0.98493	0.00000	-0.00355	0.17173	0.00000

Fock matrix (orbital energies)

-35.83281	-10.32503	-10.10889	-10.10889	0.03437	2.40056	2.40056	5.80529
-----------	-----------	-----------	-----------	---------	---------	---------	---------

one-electron matrix - mo basis

-135.65305	-3.04604	0.00000	0.00000	-0.39346	0.00000	0.00000	-12.56353
-3.04604	-109.75599	0.00000	0.00000	3.54891	0.00000	0.00000	9.29519
0.00000	0.00000	-109.85947	0.00000	0.00000	11.08280	0.00008	0.00000
0.00000	0.00000	0.00000	-109.85947	0.00000	0.00008	-11.08280	0.00000
-0.39346	3.54891	0.00000	0.00000	-35.79604	0.00000	0.00000	-7.54312
0.00000	0.00000	11.08280	0.00008	0.00000	-41.44875	0.00000	0.00000
0.00000	0.00000	0.00008	-11.08280	0.00000	0.00000	-41.44875	0.00000
-12.56353	9.29519	0.00000	0.00000	-7.54312	0.00000	0.00000	-60.54025

two-electron j-integrals

15.73639	16.06217	16.24387	16.24387	4.55934	5.62432	5.62432	8.71600
16.06217	16.07260	14.31232	14.31232	4.49908	5.49161	5.49161	8.40955
16.24387	14.31232	16.04405	14.34742	4.47177	5.57902	5.49962	8.44589
16.24387	14.31232	14.34742	16.04405	4.47177	5.49962	5.57902	8.44589
4.55934	4.49908	4.47177	4.47177	7.06993	5.18916	5.18916	5.06889
5.62432	5.49161	5.57902	5.49962	5.18916	5.02500	4.57048	4.94051
5.62432	5.49161	5.49962	5.57902	5.18916	4.57048	5.02500	4.94051
8.71600	8.40955	8.44589	8.44589	5.06889	4.94051	4.94051	6.52704

two-electron k-integrals

15.73639	4.26325	4.37635	4.37635	0.05165	0.13586	0.13586	0.63131
4.26325	16.07260	0.87601	0.87601	0.07011	0.05128	0.05128	0.42930
4.37635	0.87601	16.04405	0.84832	0.02589	0.32048	0.03221	0.31427

Appendix F: .orb output file for LiF

4.37635	0.87601	0.84832	16.04405	0.02589	0.03221	0.32048	0.31427
0.05165	0.07011	0.02589	0.02589	7.06993	0.66923	0.66923	0.47641
0.13586	0.05128	0.32048	0.03221	0.66923	5.02500	0.22726	0.35111
0.13586	0.05128	0.03221	0.32048	0.66923	0.22726	5.02500	0.35111
0.63131	0.42930	0.31427	0.31427	0.47641	0.35111	0.35111	6.52704
triple (su kk)							
15.73639	16.06217	16.24387	16.24387	4.55934	5.62432	5.62432	8.71600
0.01671	1.43161	0.61350	0.61350	-0.41043	-0.36289	-0.36289	-0.89671
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.10687	0.02341	0.09273	0.09273	-0.11005	-0.06277	-0.06277	-0.01925
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2.22673	2.05195	2.28436	2.28436	-0.06786	0.29781	0.29781	0.90747
0.01671	1.43161	0.61350	0.61350	-0.41043	-0.36289	-0.36289	-0.89671
16.06217	16.07260	14.31232	14.31232	4.49908	5.49161	5.49161	8.40955
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
-0.63300	-0.59890	-0.50445	-0.50445	0.22102	0.04900	0.04900	-0.10997
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
-1.80903	-1.45597	-1.29098	-1.29098	0.03983	-0.22553	-0.22553	-0.71200
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
16.24387	14.31232	16.04405	14.34742	4.47177	5.57902	5.49962	8.44589
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
-1.90961	-1.58614	-1.88183	-1.59101	0.12900	-0.09962	-0.16708	-0.63034
-0.00001	-0.00001	-0.00001	-0.00001	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
16.24387	14.31232	14.34742	16.04405	4.47177	5.49962	5.57902	8.44589
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
-0.00001	-0.00001	-0.00001	-0.00001	0.00000	0.00000	0.00000	0.00000
1.90961	1.58614	1.59101	1.88183	-0.12900	0.16708	0.09962	0.63034
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.10687	0.02341	0.09273	0.09273	-0.11005	-0.06277	-0.06277	-0.01925
-0.63300	-0.59890	-0.50445	-0.50445	0.22102	0.04900	0.04900	-0.10997
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
4.55934	4.49908	4.47177	4.47177	7.06993	5.18916	5.18916	5.06889
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

Appendix F: .orb output file for LiF

1.02981	0.98018	0.95763	0.95763	0.21690	0.38758	0.38758	0.77077
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
-1.90961	-1.58614	-1.88183	-1.59101	0.12900	-0.09962	-0.16708	-0.63034
-0.00001	-0.00001	-0.00001	-0.00001	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
5.62432	5.49161	5.57902	5.49962	5.18916	5.02500	4.57048	4.94051
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
-0.00001	-0.00001	-0.00001	-0.00001	0.00000	0.00000	0.00000	0.00000
1.90961	1.58614	1.59101	1.88183	-0.12900	0.16708	0.09962	0.63034
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
5.62432	5.49161	5.49962	5.57902	5.18916	4.57048	5.02500	4.94051
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2.22673	2.05195	2.28436	2.28436	-0.06786	0.29781	0.29781	0.90747
-1.80903	-1.45597	-1.29098	-1.29098	0.03983	-0.22553	-0.22553	-0.71200
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1.02981	0.98018	0.95763	0.95763	0.21690	0.38758	0.38758	0.77077
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
8.71600	8.40955	8.44589	8.44589	5.06889	4.94051	4.94051	6.52704

Appendix F: .orb output file for LiF

Appendix G: Program Listing of Analyse

```

*
* Main program
*
  program analyse
  integer iorbital,lnblnk,i,N
  parameter (N = 7)
  double precision energy(N),one(7,N),jay(7,7,N),kay(7,7,N)
  double precision fock(7,N),fockc(7,N)
  character*8 flname(N)
  character*80 root

*
  call getarg (1,root)
  i = lnblnk(root)
  open (unit=20,file=root(1:i)//'.output')
  open (unit=30,file=root(1:i)//'.orbitals')

*
  call setup (flname)
  call readenergy (root(1:i),flname,energy,iorbital)
  call readint (root(1:i),flname,one,jay,kay,fock)
  call calcfock (fock,fockc,one,jay,kay,N,iorbital)
  call writeout (flname,energy,one,jay,kay,iorbital,fock,fockc)
  call output(root(1:i),energy,one,jay,kay,iorbital,fock,fockc)

*
  end

*
*
*
*
  subroutine calcfock (fock,fockc,one,jay,kay,N,iorbital)
  integer lp,i,j,iorbital,N
  double precision one(7,*),jay(7,7,*),kay(7,7,*)
  double precision fock(7,*),fockc(7,*)

*
  do lp = 1,N
    do i = 1,7
      fockc(i,lp) = one(i,lp)
      if (lp.gt.4) then
        do j = 1,iorbital
          fockc(i,lp)=fockc(i,lp)+2.0d0*jay(i,j,lp)-kay(i,j,lp)
        enddo
      else
        fockc(i,lp)=fockc(i,lp)+2.0d0*jay(i,1,lp)-kay(i,1,lp)
      endif
    enddo
  enddo
enddo

```

```

return
end

*
*
*
subroutine calctab(table,grnd,singexc,tripexc,excited,mopace)
double precision excited(2),table(9),mopace(2),grnd,singexc
double precision tripexc

*
table(1) = grnd
table(2) = grnd + singexc
table(3) = grnd + tripexc
table(4) = singexc
table(5) = tripexc
table(6) = 100.0d0 * (singexc - mopace(1)) /mopace(1)
table(7) = 100.0d0 * (tripexc - mopace(2)) /mopace(2)
if(excited(1).ne.0.0d0)
&   table(8) = 100.0d0 * (singexc - excited(1)) /excited(1)
if(excited(2).ne.0.0d0)
&   table(9) = 100.0d0 * (tripexc - excited(2)) /excited(2)

*
return
end

*
*
*
subroutineoutput(root,energy,one,jay,kay,iorbital,fock,fockc)
character*(*) root
double precision energy(*),one(7,*),jay(7,7,*),kay(7,7,*)
double precision fock(7,*),singexc,tripexc,ground,fockc(7,*)
double precision excit(2),table(9,4),mopex(2),tabl2(9,4)
integer iorbital,loop,line,i
character*(24) text(9)

*
call readreal (root,excit)

*
text(1) = 'Ground state energy '
text(2) = 'Singlet state energy '
text(3) = 'Triplet state energy '
text(4) = '1st excitation energy '
text(5) = '2nd excitation energy '
text(6) = '1st error wrt Mopac '
text(7) = '2nd error wrt Mopac '
text(8) = '1st error wrt experiment'
text(9) = '2nd error wrt experiment'

*
mopex(1) = energy(3)-energy(2)
mopex(2) = energy(4)-energy(2)

```

Appendix G: Program Listing of Analyse

```

call calctab(table(1,1),energy(2),mopex(1),mopex(2),excit,mopex)
ground = energy(2)
do loop = 2,4
  tripexc = fockc(2,loop) - fockc(1,loop) - jay(1,2,loop)
  singexc = tripexc + 2.0d0 * kay(1,2,loop)
  call calctab(table(1,loop),ground,singexc,tripexc,excit,mopex)
enddo

*

mopex(1) = energy(6)-energy(5)
mopex(2) = energy(7)-energy(5)
call calctab(tabl2(1,1),energy(5),mopex(1),mopex(2),excit,mopex)
ground = energy(5)
do loop = 5,7
  tripexc = fockc(iorbital+1,loop)
&      - fockc(iorbital,loop) - jay(iorbital,iorbital+1,loop)
  singexc = tripexc + 2.0d0 * kay(iorbital,iorbital+1,loop)
  call calctab(tabl2(1,loop-3),ground,singexc,tripexc,excit,mopex)
enddo

*

do line = 6,20,14
  write (line,*) 'Output for ',root
  write (line,*)
  write (line,*)
  write (line,'(a,f12.5)') 'SCF ground state energy = ',energy(1)
  write (line,'(a,f12.5)') 'Singlet excitation energy = ',excit(1)
  write (line,'(a,f12.5)') 'Triplet excitation energy = ',excit(2)
  write (line,*)
  write (line,*)

*

  write (line,9990) 'CI = 2 case ', 'Mopac', 'Ground', 'Singlet',
&      'Triplet'
  write (line,*)
  do loop = 1,5
    write (line,9980) text(loop),(table(loop,i),i=1,4)
  enddo
  do loop = 6,9
    write (line,9970) text(loop),(table(loop,i),i=1,4)
  enddo
  write (line,*)
  write (line,*)

*

  write (line,9990) 'CI = lots case', 'Mopac', 'Ground', 'Singlet',
&      'Triplet'
  write (line,*)
  do loop = 1,5
    write (line,9980) text(loop),(tabl2(loop,i),i=1,4)
  enddo
  do loop = 6,9

```

Appendix G: Program Listing of Analyse

```

        write (line,9970) text(loop),(tab12(loop,i),i=1,4)
    enddo
enddo
*
9990 format (a14,14x,a7,3x,3(6x,a7))
9980 format (a22,2x,f13.5,2x,3f13.5)
9970 format (a24,f13.5,'      return
    end
*
*
*
    subroutine readenergy (root,flname,energy,iorbital)
    integer loop,index,iorbital
    character*(*) flname(*),root
    character*80 line
    double precision energy(*)
*
* Read in number of occupied orbitals in ground state
*
    open (unit=10,file=root//'_ '//flname(1)//'.out')
10 read (10,'(a80)') line
    if (index(line,'doubly occupied').eq.0) goto 10
    read (line(60:61),'(i2)') iorbital
    close (10)
    if (iorbital.ge.4) iorbital = 4
*
* Read in the energies form each file
*
    do loop = 1,7
        open (unit=10,file=root//'_ '//flname(loop)//'.out')
20 read (10,'(a80)') line
        if (index(line,'electronic energy').eq.0) goto 20
        read (line(37:52),'(f16.5)') energy(loop)
        close (10)
    enddo
    return
end
*
*
*
    subroutine readint (root,flname,one,jay,kay,fock)
    integer loop,i,j,num
    character*(*) flname(*),root
    double precision one(7,*),jay(7,7,*),kay(7,7,*),fock(7,*)
    do loop = 2,7
        open (unit=10,file=root//'_ '//flname(loop)//'.int')
        read (10,*) num
        read (10,*)

```

Appendix G: Program Listing of Analyse

```

      read (10,*) (one(i,loop),i=1,num)
      read (10,*) (fock(i,loop),i=1,num)
      read (10,*)
      do j =1,num
        read (10,*) (jay(j,i,loop),i=1,num)
      enddo
      read (10,*)
      do j =1,num
        read (10,*) (kay(j,i,loop),i=1,num)
      enddo
      close (10)
    enddo
  return
end

*
*
*
  subroutine readreal (root,excited)
    integer i
    character*(*) root
    double precision excited(*)
*
* Read in the real excitation energies (either experimental or ab-initio)
*
    i = index(root,'_') - 1
    open (unit=10,file=root(1:i)//'.eng')
    read (10,*) excited(1)
    read (10,*) excited(2)
    close (10)
    return
  end

*
*
*
  subroutine setup (fname)
    character*(*) fname(*)
    fname(1) = 'scf_grnd'
    fname(2) = 'ci2_grnd'
    fname(3) = 'ci2_sing'
    fname(4) = 'ci2_trip'
    fname(5) = 'cil_grnd'
    fname(6) = 'cil_sing'
    fname(7) = 'cil_trip'
    return
  end

*
*
*
```

Appendix G: Program Listing of Analyse

```

subroutine writeout(flname,energy,one,jay,kay,iorbital,fock,fockc)
integer loop,i,j,iorbital
character*(*) flname(*)
double precision energy(*),one(7,*),jay(7,7,*),kay(7,7,*)
double precision fock(7,*),fockc(7,*)

*
do loop = 1,7
  write (30,8) flname(loop),iorbital,energy(loop)
  write (30,*) 'Fock electron integral'
  write (30,9) (fock(i,loop),i=1,7)
  write (30,*) 'Calculated Fock integral'
  write (30,9) (fockc(i,loop),i=1,7)
  write (30,*) 'One electron integral'
  write (30,9) (one(i,loop),i=1,7)
  write (30,*) 'Coulomb integral'
  do j =1,7
    write (30,9) (jay(j,i,loop),i=1,7)
  enddo
  write (30,*) 'Exchange integral'
  do j =1,7
    write (30,9) (kay(j,i,loop),i=1,7)
  enddo
  write (30,*)
enddo

*
8 format (a,i5,f12.5)
9 format (7f10.5)
return
end

```

Appendix H: Analyse Output for Water by the AM1 Method

Output for water_aml

SCF ground state energy = -491.46848
 Singlet excitation energy = 8.69620
 Triplet excitation energy = 7.90110

CI = 2 case	Mopac	Ground	Singlet	Triplet
Ground state energy	-491.48587	-491.48587	-491.48587	-491.48587
Singlet state energy	-485.17506	-485.11345	-485.34689	-485.34689
Triplet state energy	-486.18212	-486.15035	-486.35395	-486.35395
1st excitation energy	6.31081	6.37242	6.13898	6.13898
2nd excitation energy	5.30375	5.33552	5.13192	5.13192
1st error wrt Mopac	0.00000%	0.97626%	-2.72279%	-2.72279%
2nd error wrt Mopac	0.00000%	0.59901%	-3.23978%	-3.23978%
1st error wrt experiment	-27.43026%	-26.72179%	-29.40618%	-29.40618%
2nd error wrt experiment	-32.87327%	-32.47117%	-35.04803%	-35.04803%

Appendix I: Program Listing of Charges

```

program charges
integer MAXORB,MAXAT,ln,numat,numorbs,loop,nofill,lnblnk
parameter (MAXORB = 100, MAXAT = 20)
integer iatom(MAXORB),lfirst(MAXAT),llast(MAXAT)
double precision realcharge(4,MAXAT),orbitals(4,MAXORB,MAXORB)
double precision corrcharge(3,4,MAXAT),calccharge(3,4,MAXAT)
double precision corraopop(3,4,MAXORB),mopop(2,MAXORB)
double precision calcaopop(3,4,MAXORB),errors(3,4,MAXAT)
character*80 root,name(4)
character*4 atype(MAXAT)

*
call getarg (1,root)
ln = lnblnk(root)

*
open (unit=20,file=root(1:ln)//'.charges')

*
name(1) = root(1:ln)//'_ci2_grnd'
name(2) = root(1:ln)//'_ci2_sing'
name(3) = root(1:ln)//'_cil_grnd'
name(4) = root(1:ln)//'_cil_sing'
ln = ln + 9

*
call setup(iatom,corraopop,calcaopop,mopop,realcharge,corrcharge
& ,calccharge,orbitals,MAXORB,MAXAT,lfirst,llast,errors)

*
call readdat(numat,lfirst,llast,nofill,MAXORB,MAXAT,name(1)(1:ln))

*
do loop = 1,4
  call readorb(numorbs,orbitals,MAXORB,name(loop)(1:ln),loop)
  call readoutput(realcharge,MAXAT,numat,atype
& ,name(loop)(1:ln),loop)
enddo

*
call makecharge(numorbs,numat,orbitals,iatom,corraopop,realcharge
& ,mopop,corrcharge,calccharge,lfirst,llast,MAXORB,MAXAT,atype
& ,nofill,calcaopop)
call calcerror(numat,corrcharge,realcharge,calccharge,errors)
call output(numorbs,numat,orbitals,iatom,corraopop,realcharge
& ,mopop,corrcharge,calccharge,lfirst,llast,MAXORB,MAXAT
& ,atype,nofill,6,calcaopop,errors)
call output(numorbs,numat,orbitals,iatom,corraopop,realcharge
& ,mopop,corrcharge,calccharge,lfirst,llast,MAXORB,MAXAT
& ,atype,nofill,20,calcaopop,errors)

*

```



```

end

*
*
*
subroutine calcerror(numat,corrcharge,realcharge,calccharge,errors)
integer numat,i,j,k,ihomo,ilumo
double precision realcharge(4,*),errors(3,4,*)
double precision corrcharge(3,4,*),calccharge(3,4,*)
double precision right,calc,temp
*
do i = 1,3
  do j = 1,4
    do k = 1,numat
      temp = corrcharge(i,j,k) - calccharge(i,j,k)
      errors(i,j,k) = (100.0d0 * temp) / corrcharge(i,j,k)
    enddo
  enddo
enddo
*
return
end
*
*
*
subroutine makecharge(numorbs,numat,orbitals,iatom,mol,
& realcharge,mopop,corrcharge,calccharge,lfirst,llast,MAXORB,MAXAT
& ,atype,nofill,mo2)
integer MAXORB,MAXAT,numorbs,numat,nofill,i,j,k,l
integer iatom(MAXORB),lfirst(MAXAT),llast(MAXAT)
double precision realcharge(4,MAXAT),nuccharge
double precision corrcharge(3,4,MAXAT),calccharge(3,4,MAXAT)
double precision orbitals(4,MAXORB,MAXORB),mopop(2,MAXORB)
double precision mol(3,4,MAXORB),mo2(3,4,MAXORB)
double precision corratio,valratio
character*4 atype(MAXAT)
*
* Set up the array of which orbital belong to which atom
*
do i = 1,numorbs
  do j = 1,numat
    if ((i.ge.lfirst(j)).and.(i.le.llast(j))) iatom(i) = j
  enddo
enddo
*
* Set up the two MO arrays
*
do i = 1,nofill
  mopop(1,i) = 2.0d0

```

Appendix I: Program Listing of Charges

From: J.M.Hutson@durham.ac.uk Mon Jun 10 11:22:45 1996
Date: Mon, 10 Jun 1996 09:08:40 +0100 (BST)
From: J.M.Hutson@durham.ac.uk
To: Rob Bryan <Robert.Bryan@durham.ac.uk>
Subject: Re: Simulated annealing codes (fwd)

Rob,

Here's a preliminary reply from SuYan.

Jeremy

Forwarded message:

> From: sliu@trex.uchicago.edu Sat Jun 8 02:44 BST 1996
> Date: Fri, 7 Jun 1996 20:44:13 -0500 (CDT)
> From: Suyan Liu <sliu@trex.uchicago.edu>
> Message-Id: <199606080144.UAA03926@trex.uchicago.edu>
> To: J.M.Hutson@durham.ac.uk
> Subject: Re: Simulated annealing codes
> Content-Type: text
> Content-Length: 1749

> Dear Jeremy:

> Sorry for this late response. I just began a new job (financial modeling)
> this week and it kept me occupied all the time. Since I got this offer
> very accidentally and quickly started right after, I didn't have time
> even to tell Zlatko and Jules. I will try to call them next week.
> I will prepare the codes you wanted, I only have the simulated annealing
> code for Ar_nH_20 but the newton's method and the Cerjan-Miller eigenvector
> following code (newton's for minima and Cerjan-Miller's for saddles in one
> code) and the 5D ArnHF (should be ease to replace HF by other diatoms)
> code are for Arn_HF.

> I might not be able to put detailed comments for my code, but I will be glad
> to help if your students have further questions.

> Since there are too many files even though they are not quite big still
> not convenient to send them by email. Your student can telnet to Zlatko's
> machine at NYU, with my user name "sliu" and the password "yh117623"
> the machine name is "zlatko.chem.nyu.edu". Or he can ftp through these
> informations. (better ftp)

> the paths are:

> /u/sliu/WORK/5Dprogram for 5D Ar_nHF
> /u/sliu/AUTO for Ar_nHF newton's method and cerjan-miller'
> /u/sliu/AUTO/sim_arh2o for simulated annealing Ar_nH_20

> I put all the necessary files in each directory.
> tape5 will be the read in files for all tasks (differ each of course).

> wv20.f is the main program for 5D Ar_nHF, (I calculated the wavefunctions, too)
> t.f is the main program for newton's and cerjan-miller's method
> sa.f is the main and the only one program for simulated annealing.

> For simulated, sometimes it might need to try more initial sets of
> coordinates and adjust some of the step of other parameters.

> I hope this will help.

> Best wishes:

> Suyan

r. J. M. Hutson

```

        mpopop(2,i) = 2.0d0
    enddo
    mpopop(2,nofill) = 1.0d0
    mpopop(2,nofill+1) = 1.0d0
*
* Calculate the MO arrays
*
    do i = 0,2,2
        do j = 1,2
            do k = 1,numorbs
                do l = 1,nofill-1
                    mo1(1,i+j,k) = mo1(1,i+j,k) + mpopop(j,l) * orbitals(i+j,k,l)**2
                    mo2(1,i+j,k) = mo2(1,i+j,k) + mpopop(j,l) * orbitals(i+3-j,k,l)**2
                enddo
                do l = nofill,numorbs
                    mo1(2,i+j,k) = mo1(2,i+j,k) + mpopop(j,l) * orbitals(i+j,k,l)**2
                    mo2(2,i+j,k) = mo2(2,i+j,k) + mpopop(j,l) * orbitals(i+3-j,k,l)**2
                enddo
                do l = 1,numorbs
                    mo1(3,i+j,k) = mo1(3,i+j,k) + mpopop(j,l) * orbitals(i+j,k,l)**2
                    mo2(3,i+j,k) = mo2(3,i+j,k) + mpopop(j,l) * orbitals(i+3-j,k,l)**2
                enddo
            enddo
        enddo
    enddo
*
* Calculate the total, core & active charge
* From both the calculated MO's and the correct MO's
*
    do i = 1,4
        do k = 1,3
            do j = 1,numorbs
                corrcharge(k,i,iatom(j)) = corrcharge(k,i,iatom(j)) + mo1(k,i,j)
                calccharge(k,i,iatom(j)) = calccharge(k,i,iatom(j)) + mo2(k,i,j)
            enddo
        enddo
    enddo
*
* Account for the nuclear charge
*
    corratio = (dble(nofill) - 1.0d0) / dble(nofill)
    valratio = 1.0d0 / dble(nofill)
    do i = 1,4
        do j = 1,numat
            if (atype(j).eq.'h') nuccharge = 1.0d0
            if (atype(j).eq.'li') nuccharge = 1.0d0
            if (atype(j).eq.'o') nuccharge = 6.0d0
            if (atype(j).eq.'c') nuccharge = 4.0d0

```

Appendix I: Program Listing of Charges

```

    corrcharge(1,i,j) = (nuccharge*corratio) - corrcharge(1,i,j)
    corrcharge(2,i,j) = (nuccharge*valratio) - corrcharge(2,i,j)
    corrcharge(3,i,j) = nuccharge - corrcharge(3,i,j)
    calccharge(1,i,j) = (nuccharge*corratio) - calccharge(1,i,j)
    calccharge(2,i,j) = (nuccharge*valratio) - calccharge(2,i,j)
    calccharge(3,i,j) = nuccharge - calccharge(3,i,j)
  enddo
enddo
*
return
end
*
*
*
subroutine output(numorbs,numat,orbitals,iatom,corraopop
& ,realcharge,mopop,corrcharge,calccharge,lfirst,llast,MAXORB,MAXAT
& ,atype,nofill,line,calcaopop,errors)
  integer MAXORB,MAXAT,nofill,numorbs,numat,i,j,line
  integer iatom(MAXORB),lfirst(MAXAT),llast(MAXAT)
  double precision realcharge(4,MAXAT),orbitals(4,MAXORB,MAXORB)
  double precision corrcharge(3,4,MAXAT),calccharge(3,4,MAXAT)
  double precision corraopop(3,4,MAXORB),mopop(2,MAXORB)
  double precision calcaopop(3,4,MAXORB),errors(3,4,MAXAT)
  character*4 atype(MAXAT)
*
  write (line,*) 'Number of filled orbitals',nofill
  write (line,*)
*
  write (line,*) 'Atom      Element      First Orbital'//
&          '      Last Orbital'
  do i = 1,numat
    write(line,'(i3,i1x,a4,2(i15,3x))')i,atype(i),lfirst(i),llast(i)
  enddo
  write (line,*)
*
  write (line,*) 'Orbital      Atom      '//'
&          'Ground      Excited'
  do i = 1,numorbs
    write(line,'(i4,i15,10x,2f16.5)')i,iatom(i),mopop(1,i),mopop(2,i)
  enddo
  write (line,*)
*
  write (line,*) 'Real total charges - read from the output'
  write (line,*) 'Atom      CI2 Ground      CI2 Excited'//
&          '      CI7 Ground      CI7 Excited'
  do i = 1,numat
    write (line,9990) i,(realcharge(j,i),j=1,4)
  enddo

```

Appendix I: Program Listing of Charges

```

write (line,*)
*
* Set of own orbitals and corresponding charges
*
write (line,*) 'own MOs - over core orbitals'
write (line,*) 'Orbital   CI2 Ground   CI2 Excited'//
&           '          CI7 Ground   CI7 Excited'
do i = 1,numorbs
  write (line,9990) i,(corraopop(1,j,i),j=1,4)
enddo
write (line,*)
*
write (line,*) 'Core charges - from own MOs'
write (line,*) 'Atom      CI2 Ground   CI2 Excited'//
&           '          CI7 Ground   CI7 Excited'
do i = 1,numat
  write (line,9990) i,(corrcharge(1,j,i),j=1,4)
enddo
write (line,*)
*
write (line,*) 'own MOs - over active orbitals'
write (line,*) 'Orbital   CI2 Ground   CI2 Excited'//
&           '          CI7 Ground   CI7 Excited'
do i = 1,numorbs
  write (line,9990) i,(corraopop(2,j,i),j=1,4)
enddo
write (line,*)
*
write (line,*) 'Active charges - from own MOs'
write (line,*) 'Atom      CI2 Ground   CI2 Excited'//
&           '          CI7 Ground   CI7 Excited'
do i = 1,numat
  write (line,9990) i,(corrcharge(2,j,i),j=1,4)
enddo
write (line,*)
*
write (line,*) 'own MOs - over all orbitals'
write (line,*) 'Orbital   CI2 Ground   CI2 Excited'//
&           '          CI7 Ground   CI7 Excited'
do i = 1,numorbs
  write (line,9990) i,(corraopop(3,j,i),j=1,4)
enddo
write (line,*)
*
write (line,*) 'Total charges - from own MOs'
write (line,*) 'Atom      CI2 Ground   CI2 Excited'//
&           '          CI7 Ground   CI7 Excited'
do i = 1,numat

```

Appendix I: Program Listing of Charges

```

        write (line,9990) i,(corrcharge(3,j,i),j=1,4)
    enddo
    write (line,*)
*
*
* Set of other orbital and corresponding charges
*
*
    write (line,*) 'other MOs - over core orbitals'
    write (line,*) 'Orbital   CI2 Ground   CI2 Excited'//
&          '          CI7 Ground   CI7 Excited'
    do i = 1,numorbs
        write (line,9990) i,(calcaopop(1,j,i),j=1,4)
    enddo
    write (line,*)
*
    write (line,*) 'Core charges - from other MOs'
    write (line,*) 'Atom       CI2 Ground   CI2 Excited'//
&          '          CI7 Ground   CI7 Excited'
    do i = 1,numat
        write (line,9990) i,(calccharge(1,j,i),j=1,4)
    enddo
    write (line,*)
*
    write (line,*) 'other MOs - over active orbitals'
    write (line,*) 'Orbital   CI2 Ground   CI2 Excited'//
&          '          CI7 Ground   CI7 Excited'
    do i = 1,numorbs
        write (line,9990) i,(calcaopop(2,j,i),j=1,4)
    enddo
    write (line,*)
*
    write (line,*) 'Active charges - from other MOs'
    write (line,*) 'Atom       CI2 Ground   CI2 Excited'//
&          '          CI7 Ground   CI7 Excited'
    do i = 1,numat
        write (line,9990) i,(calccharge(2,j,i),j=1,4)
    enddo
    write (line,*)
*
    write (line,*) 'other MOs - over all orbitals'
    write (line,*) 'Orbital   CI2 Ground   CI2 Excited'//
&          '          CI7 Ground   CI7 Excited'
    do i = 1,numorbs
        write (line,9990) i,(corraopop(3,j,i),j=1,4)
    enddo
    write (line,*)
*

```

Appendix I: Program Listing of Charges

```

write (line,*) 'Total charges - from other MOs'
write (line,*) 'Atom      CI2 Ground      CI2 Excited'//
&          '      CI7 Ground      CI7 Excited'
do i = 1,numat
  write (line,9990) i,(calccharge(3,j,i),j=1,4)
enddo
write (line,*)

*
* Errors
*
  write (line,*)
  write (line,*) 'Errors in core charges'
  write (line,*) 'Atom      CI2 Ground      CI2 Excited'//
&          '      CI7 Ground      CI7 Excited'
do i = 1,numat
  write (line,9980) i,(errors(1,j,i),j=1,4)
enddo

*
  write (line,*)
  write (line,*) 'Errors in Active charges'
  write (line,*) 'Atom      CI2 Ground      CI2 Excited'//
&          '      CI7 Ground      CI7 Excited'
do i = 1,numat
  write (line,9980) i,(errors(2,j,i),j=1,4)
enddo

*
  write (line,*)
  write (line,*) 'Errors in total charges'
  write (line,*) 'Atom      CI2 Ground      CI2 Excited'//
&          '      CI7 Ground      CI7 Excited'
do i = 1,numat
  write (line,9980) i,(errors(3,j,i),j=1,4)
enddo
write (line,*)

*
9990 format (i3,4f16.5)
9980 format (i3,4(f15.4,'*
  return
end

*
*
*
  subroutine readdat(numat,lfirst,llast,nofill,MAXORB,MAXAT,name)
  integer MAXORB,MAXAT,nofill,numat,i,lfirst(MAXAT),llast(MAXAT)
  character*(*) name

*
  open (unit=10,file=name//'.dat')
*

```

Appendix I: Program Listing of Charges

```

      read (10,*) numat
      do i = 1,numat
        read (10,*) lfirst(i),llast(i)
      enddo
      read (10,*) nofill
*
      close(10)
*
      return
      end
*
*
*
      subroutine readorb(numorbs,orbitals,MAXORB,name,loop)
      integer numorbs,MAXORB,i,j,loop
      double precision orbitals(4,MAXORB,MAXORB)
      character*(*) name
*
      open (unit=10,file=name//'.orb')
*
      read (10,*) numorbs
      do i = 1,numorbs
        do j = 1,numorbs
          read (10,*) orbitals(loop,i,j)
        enddo
      enddo
*
      close(10)
      return
      end
*
*
*
      subroutine readoutput(realcharge,MAXAT,numat,atype,name,loop)
      double precision realcharge(4,MAXAT)
      integer i,numat,MAXAT,dummy,loop
      character*80 line
      character*(*) name
      character*4 atype(MAXAT)
*
      open (unit=10,file=name//'.out')
*
10 read (10,'(a80)') line
   if (index(line,'net atomic charges').eq.0) goto 10
   read (10,'(a80)') line
   read (10,'(a80)') line
   do i = 1,numat
     read (10,*) dummy,atype(i),realcharge(loop,i)

```

Appendix I: Program Listing of Charges


```

        enddo
*
        close (10)
        return
    end
*
*
*
    subroutine setup(iatom,corraopop,calcaopop,mopop,realcharge,
& corrcharge,calccharge,orbitals,MAXORB,MAXAT,lfirst,llast,errors)
    integer MAXORB,MAXAT,iatom(MAXORB),lfirst(MAXAT),llast(MAXAT)
    double precision realcharge(4,MAXAT),orbitals(4,MAXORB,MAXORB)
    double precision corrcharge(3,4,MAXAT),calccharge(3,4,MAXAT)
    double precision mopop(2,MAXORB),corraopop(3,4,MAXORB)
    double precision calcaopop(3,4,MAXORB),errors(3,4,MAXAT)
*
    call zeroi1(iatom,MAXORB)
    call zeroi1(lfirst,MAXAT)
    call zeroi1(llast,MAXAT)
    call zeror2(realcharge,4,MAXAT)
    call zeror3(corrcharge,3,4,MAXAT)
    call zeror3(calccharge,3,4,MAXAT)
    call zeror3(orbitals,4,MAXORB,MAXORB)
    call zeror3(corraopop,3,4,MAXORB)
    call zeror3(calcaopop,3,4,MAXORB)
    call zeror3(errors,3,4,MAXAT)
    call zeror2(mopop,2,MAXORB)
*
    return
    end
*
*
*
    subroutine zeroi1(array,m)
    integer i,m,array(m)
*
    do i = 1,m
        array(i) = 0
    enddo
*
    return
    end
*
*
*
    subroutine zeror2(array,m,n)
    integer i,j,m,n
    double precision array(m,n)

```

Appendix I: Program Listing of Charges

```

*
do i = 1,m
  do j = 1,n
    array(i,j) = 0.0d0
  enddo
enddo
*
return
end
*
*
*
subroutine zeror3(array,m,n,o)
integer i,j,k,m,n,o
double precision array(m,n,o)
*
do i = 1,m
  do j = 1,n
    do k = 1,o
      array(i,j,k) = 0.0d0
    enddo
  enddo
enddo
*
return
end

```

Appendix J: Charges Output for Water by the AM1 Method

Number of filled orbitals 4

Atom	Element	First Orbital	Last Orbital
1	h	1	1
2	o	2	5
3	h	6	6

Orbital	Atom	Ground	Excited
1	1	2.00000	2.00000
2	2	2.00000	2.00000
3	2	2.00000	2.00000
4	2	2.00000	1.00000
5	2	0.00000	1.00000
6	3	0.00000	0.00000

Real total charges - read from the output

Atom	Ground	Excited
1	0.20330	-0.04080
2	-0.40670	0.08170
3	0.20330	-0.04080

own MOs - over core orbitals

Orbital	Ground	Excited
1	0.79667	0.70571
2	1.85942	1.84252
3	1.23882	1.32915
4	1.30842	1.41692
5	0.00000	0.00000
6	0.79667	0.70571

Core charges - from own MOs

Atom	Ground	Excited
1	-0.04667	0.04429
2	0.09333	-0.08858
3	-0.04667	0.04429

own MOs - over active orbitals

Orbital	Ground	Excited
1	0.00000	0.33512
2	0.00000	0.07874
3	0.00000	0.08142
4	0.00000	0.16960
5	2.00000	1.00000
6	0.00000	0.33512

Appendix J: Charges Output for Water by the AM1 Method

Active charges - from own MOs

Atom	Ground	Excited
1	0.25000	-0.08512
2	-0.50000	0.17024
3	0.25000	-0.08512

Orbital	Ground	Excited
1	0.79667	1.04083
2	1.85942	1.92126
3	1.23882	1.41057
4	1.30842	1.58651
5	2.00000	1.00000
6	0.79667	1.04083

Total charges - from own MOs

Atom	Ground	Excited
1	0.20333	-0.04083
2	-0.40667	0.08166
3	0.20333	-0.04083

other MOs - over core orbitals

Orbital	Ground	Excited
1	0.70571	0.79667
2	1.84252	1.85942
3	1.32915	1.23882
4	1.41692	1.30842
5	0.00000	0.00000
6	0.70571	0.79667

Core charges - from other MOs

Atom	Ground	Excited
1	0.04429	-0.04667
2	-0.08858	0.09333
3	0.04429	-0.04667

other MOs - over active orbitals

Orbital	Ground	Excited
1	0.00000	0.30803
2	0.00000	0.07029
3	0.00000	0.10174
4	0.00000	0.21192
5	2.00000	1.00000
6	0.00000	0.30803

Active charges - from other MOs

Atom	Ground	Excited
1	0.25000	-0.05803
2	-0.50000	0.11606
3	0.25000	-0.05803

Appendix J: Charges Output for Water by the AM1 Method

other MOs - over all orbitals

Orbital	Ground	Excited
1	0.79667	1.04083
2	1.85942	1.92126
3	1.23882	1.41057
4	1.30842	1.58651
5	2.00000	1.00000
6	0.79667	1.04083

Total charges - from other MOs

Atom	Ground	Excited
1	0.29429	-0.10470
2	-0.58858	0.20939
3	0.29429	-0.10470

Errors in core charges

Atom	Ground	Excited
1	194.9112%	205.3617%
2	194.9114%	205.3614%
3	194.9117%	205.3611%

Errors in active charges

Atom	Ground	Excited
1	0.0000%	31.8273%
2	0.0000%	31.8270%
3	0.0000%	31.8267%

Errors in total charges

Atom	Ground	Excited
1	-44.7338%	-156.4292%
2	-44.7337%	-156.4295%
3	-44.7336%	-156.4298%

Appendix K: Program Listing of Corresp

```

*
* Main program
*
      program corresp
      implicit none
      include 'Include.blk'
*
      integer i,j,m,ln,numarg,curarg
      double precision store(MAXORB,MAXORB)
      character*80 param,root
*
* Clear all the arrays
*
      call setup(iatom,lfirst,llast,charge,rhf,mopop,aopop,rhfoverlap,
& orbitals,D,DD,V,L,U,neworbitals,newD,E,energy,neweng,rhfeng,
& original,origeng)
*
* Read in command line parameters
*
      swap = .false.
      mix = .false.
*
      numarg = iargc()
      curarg = 0
10  curarg = curarg + 1
      call getarg(curarg,param)
      write (6,*) curarg,param
*
      if (param(1:2).eq.'-i') then
          curarg = curarg + 1
          call getarg(curarg,root)
          do i = 80,1,-1
              if (root(i:i).eq.' ') ln = i-1
          enddo
      endif
*
      if (param(1:2).eq.'-m') then
          mix = .true.
          curarg = curarg + 1
          call getarg(curarg,param)
          read (param,*) lambda
      endif
*
      if (curarg.ne.numarg) goto 10

```

```

*
  write (6,*) 'Input root name is ',root
  if (mix) write (6,*) 'Mixing is on, lambda =',lambda
  if (.not.mix) write (6,*) 'Mixing is off'
  write (6,*)

*
*
* Read in the orbital matrices and general info from mopac output
*
  callread(root(1:ln),numat,lfirst,llast,numfill,numorbs,rhf,rhfeng)
*   call readdat(numat,lfirst,llast,numfill,root(1:ln))
*   call readorb(numorbs,rhf,root(1:ln))
*   call readout(numorbs,rhfeng,root(1:ln))
*
* If say numorbs = numorbs + 1 then we are bracketing the whole of the
* active orbitals for both states. Therefore swap = .false.
  if (.not.swap) numfill = numfill + 1
*
* Calculate the atomic orbital population
*
  call calcpop(numfill,numorbs,mopop,aopop,rhf)
*
* Calculate the RHF overlap matrices
*
  do i = 1,2
    do j = 1,2
      call dgemm('T','N',numorbs,numorbs,numorbs,1.0d0,rhf(1,1,i),
& MAXORB,rhf(1,1,j),MAXORB,0.0d0,rhfoverlap(1,1,i,j),MAXORB)
    enddo
  enddo
*
  call header('Ground State')
  call printao(aopop(1,1),mopop(1,1),numorbs)
  call screen2('MO orbitals (RHF)',rhf(1,1,1),rhfeng(1,1),numorbs)
  call screen('Orthogonality',rhfoverlap(1,1,1,1),numorbs)
*
  call header('Excited State')
  call printao(aopop(1,2),mopop(1,2),numorbs)
  call screen2('MO orbitals (RHF)',rhf(1,1,2),rhfeng(1,2),numorbs)
  call screen('Orthogonality',rhfoverlap(1,1,2,2),numorbs)
*
  call header('Both States')
  call screen('Overlap',rhfoverlap(1,1,2,1),numorbs)
*
  call header('Corresponding Orbitals')
*
* Calculate the full alphas and betas - original
*

```

Appendix K: Program Listing of Corresp

```

call calcorig(numorbs,numfill,rhf,original,rhfeng,origeng)
call screen2('Original Ground MO (Alpha)',original(1,1,1,1),
&
& origeng(1,1,1),numorbs)
call screen2('Original Ground MO (Beta)', original(1,1,2,1),
&
& origeng(1,2,1),numorbs)
call screen2('Original Excited MO (Alpha)',original(1,1,1,2),
&
& origeng(1,1,2),numorbs)
call screen2('Original Excited MO (Beta)', original(1,1,2,2),
&
& origeng(1,2,2),numorbs)
*
* Calculate the alphas and betas
*
call calcmat(numorbs,numfill,rhf,orbitals,rhfeng,energy)
call screen2('Ground MO (Alpha)',orbitals(1,1,1,1),
&
& energy(1,1,1),numorbs)
call screen2('Ground MO (Beta)', orbitals(1,1,2,1),
&
& energy(1,2,1),numorbs)
call screen2('Excited MO (Alpha)',orbitals(1,1,1,2),
&
& energy(1,1,2),numorbs)
call screen2('Excited MO (Beta)', orbitals(1,1,2,2),
&
& energy(1,2,2),numorbs)
*
* Calculate the overlap matrix D for alpha and beta
*
do m = 1,2
call dgemm('T','N',numorbs,numorbs,numorbs,1.0d0,
&
& orbitals(1,1,m,2),MAXORB,orbitals(1,1,m,1),MAXORB,
&
& 0.0d0,D(1,1,m),MAXORB)
enddo
call screen('D (Alpha)',D(1,1,1),numorbs)
call screen('D (Beta)',D(1,1,2),numorbs)
*
* Calculate D+D for alpha and beta
*
do m= 1,2
call dgemm('T','N',numorbs,numorbs,numorbs,1.0d0,D(1,1,m),
&
& MAXORB,D(1,1,m),MAXORB,0.0d0,DD(1,1,m),MAXORB)
enddo
call screen('DdagD (Alpha)',DD(1,1,1),numorbs)
call screen('DdagD (Beta)',DD(1,1,2),numorbs)
*
* Calculate Eigenvectors and inverse root of L for alpha and beta
*
call calcvce(numfill,V,DD,L,E,gotzero)
call screen2('V (Alpha)',V(1,1,1),E(1,1),numorbs)
call screen2('V (Beta)',V(1,1,2),E(1,2),numorbs)
call screen('L (Alpha)',L(1,1,1),numorbs)
call screen('L (Beta)',L(1,1,2),numorbs)

```

Appendix K: Program Listing of Corresp


```

*
* Calculate U for alpha and beta - from D V L-(1/2)
*
  do m = 1,2
    call dgemm('N','N',numorbs,numorbs,numorbs,1.0d0,V(1,1,m),
&             MAXORB,L(1,1,m),MAXORB,0.0d0,store,MAXORB)
    call dgemm('N','N',numorbs,numorbs,numorbs,1.0d0,D(1,1,m),
&             MAXORB,store,MAXORB,0.0d0,U(1,1,m),MAXORB)
  enddo
*
* If getzero(m) is set then we need to repair U
*
  call screen('U (Alpha) - via Lambda, pre repair',U(1,1,1),numorbs)
  call screen('U (Beta) - via Lambda, pre repair',U(1,1,2),numorbs)
  if (gotzero(1).eq.1) call zeroval(numfill,numorbs,U(1,1,1),E(1,1))
  if (gotzero(2).eq.1) call zeroval(numfill,numorbs,U(1,1,2),E(1,2))
  call screen('U (Alpha) - via Lambda',U(1,1,1),numorbs)
  call screen('U (Beta) - via Lambda',U(1,1,2),numorbs)
*
* Calculate new orbitals
*
  do m = 1,2
    call dgemm('N','N',numorbs,numorbs,numorbs,1.0d0,
&             orbitals(1,1,m,1),MAXORB,V(1,1,m),MAXORB,0.0d0,
&             neworbitals(1,1,m,1),MAXORB)
    call dgemm('N','N',numorbs,numorbs,numorbs,1.0d0,
&             orbitals(1,1,m,2),MAXORB,U(1,1,m),MAXORB,0.0d0,
&             neworbitals(1,1,m,2),MAXORB)
  enddo
*
* Calculate the new orbital energies
*
  call calceng(numfill,U,V,energy,neweng)
  call screen2('New Ground MO (Alpha)',neworbitals(1,1,1,1),
&             neweng(1,1,1),numorbs)
  call screen2('New Ground MO (Beta)',neworbitals(1,1,2,1),
&             neweng(1,2,1),numorbs)
  call screen2('New Excited MO (Alpha)',neworbitals(1,1,1,2),
&             neweng(1,1,2),numorbs)
  call screen2('New Excited MO (Beta)',neworbitals(1,1,2,2),
&             neweng(1,2,2),numorbs)
*
* Mix denegenerate orbitals
*
  if (mix) call mixdeng(numfill,numorbs,neworbitals,neweng,lambda)
  call screen2('Mixed Ground MO (Alpha)',neworbitals(1,1,1,1),
&             neweng(1,1,1),numorbs)
  call screen2('Mixed Ground MO (Beta)',neworbitals(1,1,2,1),

```

Appendix K: Program Listing of Corresp

```

&                                neweng(1,2,1),numorbs)
  call screen2('Mixed Excited MO (Alpha)',neworbitals(1,1,1,2),
&                                neweng(1,1,2),numorbs)
  call screen2('Mixed Excited MO (Beta)',neworbitals(1,1,2,2),
&                                neweng(1,2,2),numorbs)
*
* Sort the orbitals into ascending orbital energy order
* But do not swap the excited beta back into place
*
  call sortorb(numfill,numorbs,neworbitals,neweng)
  call screen2('Ordered Ground MO (Alpha)',neworbitals(1,1,1,1),
&                                neweng(1,1,1),numorbs)
  call screen2('Ordered Ground MO (Beta)',neworbitals(1,1,2,1),
&                                neweng(1,2,1),numorbs)
  call screen2('Ordered Excited MO (Alpha)',neworbitals(1,1,1,2),
&                                neweng(1,1,2),numorbs)
  call screen2('Ordered Excited MO (Beta)',neworbitals(1,1,2,2),
&                                neweng(1,2,2),numorbs)
*
* Calculate the new overlap matrix D for alpha and beta - Bdag A
*
  do m = 1,2
    call dgemm('T','N',numorbs,numorbs,numorbs,1.0d0,
&            neworbitals(1,1,m,2),MAXORB,neworbitals(1,1,m,1),MAXORB,
&            0.0d0,newD(1,1,m),MAXORB)
  enddo
  call screen('New D (Alpha) - ordered',newD(1,1,1),numorbs)
  call screen('New D (Beta) - ordered',newD(1,1,2),numorbs)
*
* Now replace the empty places back in the orbital matrix
* and know about either whether we need to swap or not
*
  call replace(numfill,numorbs,neworbitals,neweng,original,origeng,
&            swap)
  call screen2('Replaced Ground MO (Alpha)',neworbitals(1,1,1,1),
&                                neweng(1,1,1),numorbs)
  call screen2('Replaced Ground MO (Beta)',neworbitals(1,1,2,1),
&                                neweng(1,2,1),numorbs)
  call screen2('Replaced Excited MO (Alpha)',neworbitals(1,1,1,2),
&                                neweng(1,1,2),numorbs)
  call screen2('Replaced Excited MO (Beta)',neworbitals(1,1,2,2),
&                                neweng(1,2,2),numorbs)
*
* Calculate the overlap for the new matrices
*
  do i = 1,2
    do j = 1,2
      call dgemm('T','N',numorbs,numorbs,numorbs,1.0d0,

```

Appendix K: Program Listing of Corresp

```

& neworbitals(1,1,i,j),MAXORB,neworbitals(1,1,i,j),MAXORB,0.0d0,
& rhfoverlap(1,1,i,j),MAXORB)
    enddo
  enddo
  call screen('Replaced Ground MO (Alpha) overlap',
&              rhfoverlap(1,1,1,1),numorbs)
  call screen('Replaced Ground MO (Beta) overlap',
&              rhfoverlap(1,1,2,1),numorbs)
  call screen('Replaced Excited MO (Alpha) overlap',
&              rhfoverlap(1,1,1,2),numorbs)
  call screen('Replaced Excited MO (Beta) overlap',
&              rhfoverlap(1,1,2,2),numorbs)
*
* Calculate the new overlap matrix D for alpha and beta - Bdag A
*
  do m = 1,2
    call dgemm('T','N',numorbs,numorbs,numorbs,1.0d0,
&              neworbitals(1,1,m,2),MAXORB,neworbitals(1,1,m,1),MAXORB,
&              0.0d0,newD(1,1,m),MAXORB)
  enddo
  call screen('New D (Alpha) - replaced',newD(1,1,1),numorbs)
  call screen('New D (Beta) - replaced',newD(1,1,2),numorbs)
*
* Output the results
*
  call writeorb(numorbs,neworbitals,root(1:ln))
*
9990 format (i5,')',100f10.6)
  end
*
*
*
  subroutine calceng(numfill,U,V,energy,neweng)
  implicit none
  include 'Include.blk'
  integer i,j,k
*
  do i = 1,2
    do j = 1,numfill
      do k = 1,numfill
        neweng(j,i,1) = neweng(j,i,1) + V(k,j,i)**2 * energy(k,i,1)
        neweng(j,i,2) = neweng(j,i,2) + U(k,j,i)**2 * energy(k,i,2)
      enddo
    enddo
  enddo
*
  return
  end

```

```

*
*
*
subroutine calcmat(numorbs,numfill,rhf,orbitals,rhfeng,energy)
implicit none
include 'Include.blk'
*
integer j,k
*
do j = 1,numfill
  energy(j,1,1) = rhfeng(j,1)
  energy(j,2,1) = rhfeng(j,1)
  do k = 1,numorbs
    orbitals(k,j,1,1) = rhf(k,j,1)
  enddo
  do k = 1,numorbs
    orbitals(k,j,2,1) = rhf(k,j,1)
  enddo
enddo
*
do j = 1,numfill-1
  energy(j,1,2) = rhfeng(j,2)
  energy(j,2,2) = rhfeng(j,2)
  do k = 1,numorbs
    orbitals(k,j,1,2) = rhf(k,j,2)
  enddo
  do k = 1,numorbs
    orbitals(k,j,2,2) = rhf(k,j,2)
  enddo
enddo
energy(numfill,1,2) = rhfeng(numfill,2)
energy(numfill,2,2) = rhfeng(numfill+1,2)
do k = 1,numorbs
  orbitals(k,numfill,1,2) = rhf(k,numfill,2)
enddo
do k = 1,numorbs
  orbitals(k,numfill,2,2) = rhf(k,numfill+1,2)
enddo
*
return
end
*
*
*
subroutine calcorig(numorbs,numfill,rhf,original,rhfeng,origeng)
implicit none
include 'Include.blk'
*

```

```

integer i,j,k,m
*
do i = 1,2
  do j = 1,2
    do k = 1,numorbs
      origeng(k,i,j) = rhfeng(k,j)
      do m = 1,numorbs
        original(m,k,i,j) = rhf(m,k,j)
      enddo
    enddo
  enddo
enddo
*
return
end
*
*
*
subroutine calcpop(numfill,numorbs,mopop,aopop,rhf)
implicit none
include 'Include.blk'
*
integer i,j,m
*
do i = 1,numfill-1
  mopop(i,1) = 2.0d0
  mopop(i,2) = 2.0d0
enddo
*
mopop(numfill,1) = 2.0d0
mopop(numfill+1,1) = 0.0d0
mopop(numfill,2) = 1.0d0
mopop(numfill+1,2) = 1.0d0
*
do i = numfill+2,numorbs
  mopop(i,1) = 0.0d0
  mopop(i,2) = 0.0d0
enddo
*
do m = 1,2
  do i = 1,numorbs
    do j = 1,numorbs
      aopop(i,m) = aopop(i,m) + rhf(i,j,m)**2 * mopop(j,m)
    enddo
  enddo
enddo
*
return

```

```

end

*
*
*
subroutine calcvee(numfill,V,DD,L,E,gotzero)
implicit none
include 'Include.blk'

*
integer ifail,i,m
double precision dum(MAXORB),r(MAXORB)
external f02abf

*
do m = 1,2
  gotzero(m) = .false.
  ifail = 0
  call f02abf(DD(1,1,m),MAXORB,numfill,E(1,m),V(1,1,m),MAXORB,
&            dum,ifail)
  if (ifail.ne.0) STOP
  do i = 1,numfill
    if (E(i,m).gt.accuracy) then
      L(i,i,m) = 1.0d0 / sqrt(E(i,m))
    else
      gotzero(m) = .true.
    endif
  enddo
enddo

*
return
end

*
*
*

subroutine getarg(n,param)
implicit none
character*(80) param
integer i,n

*
open (unit=30,file='Input',status='old',form='formatted')
do i = 1,n
  read (30,'(a80)') param
enddo
close(30)

*
return
end

*
*
*

```

```

subroutine header(string)
implicit none
character*(*) string
write (6,*) string
string = '=====',
write (6,*) string
write (6,*)

*

    return
end

*
*
*

subroutine mixdeng(numfill,numorbs,neworbitals,neweng,lambda)
implicit none
include 'Include.blk'
integer i,j,k
double precision sum

*

do i = 1,2
  do j = 1,2
    sum = neweng(3,i,j) + neweng(4,i,j)
    neweng(3,i,j) = sum * lambda
    neweng(4,i,j) = sum * (1.0d0 - lambda)
    do k = 1,numorbs
      sum = neworbitals(k,3,i,j) + neworbitals(k,4,i,j)
      neworbitals(k,3,i,j) = sum * lambda
      neworbitals(k,4,i,j) = sum * (1.0d0 - lambda)
    enddo
  enddo
enddo

*

return
end

*
*
*

subroutine printao(array1,array2,numorbs)
implicit none
include 'Include.blk'
double precision array1(MAXORB),array2(MAXORB)
integer i

*

write (6,*) 'Electron occupancy of AOs and MOs'
write (6,*)
write (6,*) '  No      AO      MO'
do i = 1,numorbs
  write (6,9990) i,array1(i),array2(i)

```

```

        enddo
        write (6,*)
*
9990 format (i5,')',100f10.6)
        return
        end
*
*
*
        subroutineread(root,numat,lfirst,llast,numfill,numorbs,rhf,rhfeng)
        implicit none
        include 'Include.blk'
*
        character*(*) root
        character*(80) filename
*
        integer idum,i,j
*
        filename = root//'_grnd.bin'
        open (unit=14,file=filename,status='old',form='unformatted')
        filename = root//'_sing.bin'
        open (unit=15,file=filename,status='old',form='unformatted')
*
        read (14)
*
        read (14) numat
        read (14) numorbs
        read (14) numfill
        read (14)
        read (14)
        do i = 1,numat
            read (14) idum,lfirst(i),llast(i)
        enddo
*
        do i = 1,numorbs
            read (14) (rhf(i,j,1),j=1,numorbs)
        enddo
        read (14) (rhfeng(i,1),i=1,numorbs)
*
        read (15)
        read (15)
        read (15)
        read (15)
        read (15)
        read (15)
        read (15)
        read (15)
        do i = 1,numorbs

```



```

        read (15) (rhf(i,j,2),j=1,numorbs)
    enddo
*
    read (15) (rhfeng(i,2),i=1,numorbs)
*
    close(14)
    close(15)
    return
end

*
*
*
    subroutine replace(numfill,numorbs,neworbitals,neweng,original,
&    origeng,swap)
    implicit none
    include 'Include.blk'
    integer i,j,k,m
*
* Need to know about swap
*
    if (swap) then
*
* First swap the Excited beta back into place
*
        neweng(numfill+1,2,2) = neweng(numfill,2,2)
        neweng(numfill,2,2) = 0.0d0
        do i = 1,numorbs
            neworbitals(i,numfill+1,2,2) = neworbitals(i,numfill,2,2)
            neworbitals(i,numfill,2,2) = 0.0d0
        enddo
*
        do i = numfill+2,numorbs
            neweng(i,1,1) = origeng(i,1,1)
            neweng(i,2,1) = origeng(i,2,1)
            neweng(i,1,2) = origeng(i,1,2)
            neweng(i,2,2) = origeng(i,2,2)
            do k = 1,numorbs
                neworbitals(k,i,1,1) = original(k,i,1,1)
                neworbitals(k,i,2,1) = original(k,i,2,1)
                neworbitals(k,i,1,2) = original(k,i,1,2)
                neworbitals(k,i,2,2) = original(k,i,2,2)
            enddo
        enddo
        neweng(numfill+1,1,1) = origeng(numfill+1,1,1)
        neweng(numfill+1,2,1) = origeng(numfill+1,2,1)
        neweng(numfill+1,1,2) = origeng(numfill+1,1,2)
        neweng(numfill,2,2) = origeng(numfill,2,2)
        do k = 1,numorbs

```

Appendix K: Program Listing of Corresp

```

        neworbitals(k,numfill+1,1,1) = original(k,numfill+1,1,1)
        neworbitals(k,numfill+1,2,1) = original(k,numfill+1,2,1)
        neworbitals(k,numfill+1,1,2) = original(k,numfill+1,1,2)
        neworbitals(k,numfill,2,2)   = original(k,numfill,2,2)
    enddo
else
    do i = numfill+1,numorbs
        do j = 1,2
            do k = 1,2
                neweng(i,j,k) = origeng(i,j,k)
                do m = 1,numorbs
                    neworbitals(m,i,j,k) = original(m,i,j,k)
                enddo
            enddo
        enddo
    enddo
endif
*
return
end
*
*
*
subroutine screen(string,matrix,numorbs)
implicit none
include 'Include.blk'
*
integer i,j
double precision matrix(MAXORB,MAXORB)
character*(*) string
*
write (6,*) string
write (6,*)
do i = 1,numorbs
    write (6,9980) (matrix(i,j),j=1,numorbs)
enddo
write (6,*)
*
9980 format (5x,100f18.12)
*
return
end
*
*
*
subroutine screen2(string,matrix,matrix2,numorbs)
implicit none
include 'Include.blk'

```

```

*
integer i,j
double precision matrix(MAXORB,MAXORB),matrix2(MAXORB)
character*(*) string
*
write (6,*) string
write (6,*)
write (6,9980) (matrix2(i),i=1,numorbs)
write (6,*)
do i = 1,numorbs
  write (6,9980) (matrix(i,j),j=1,numorbs)
enddo
write (6,*)
*
9980 format (5x,100f18.12)
*
return
end
*
*
*
subroutine setup(iatom,lfirst,llast,charge,rhf,mopop,aopop,
& rhfoverlap,orbitals,D,V,L,U,neworbitals,newD,E,energy,neweng,
& rhfeng,original,origeng)
implicit none
include 'Include.blk'
*
call zeroi1(iatom,MAXORB)
call zeroi1(lfirst,MAXAT)
call zeroi1(llast,MAXAT)
call zeror2(charge,MAXAT,2)
call zeror2(mopop,MAXORB,2)
call zeror2(aopop,MAXORB,2)
*
call zeror3(rhf,MAXORB,MAXORB,2)
call zeror4(rhfoverlap,MAXORB,MAXORB,2,2)
*
call zeror4(orbitals,MAXORB,MAXORB,2,2)
call zeror3(D,MAXORB,MAXORB,2)
call zeror3(DD,MAXORB,MAXORB,2)
call zeror3(V,MAXORB,MAXORB,2)
call zeror3(L,MAXORB,MAXORB,2)
call zeror3(U,MAXORB,MAXORB,2)
call zeror4(neworbitals,MAXORB,MAXORB,2,2)
call zeror3(newD,MAXORB,MAXORB,2)
call zeror2(E,MAXORB,2)
call zeror3(energy,MAXORB,2,2)
call zeror3(neweng,MAXORB,2,2)

```

Appendix K: Program Listing of Corresp

```

call zeror2(rhfeng,MAXORB,2)
call zeror4(original,MAXORB,MAXORB,2,2)
call zeror3(origeng,MAXORB,MAXORB,2)
*
*
return
end
*
*
*
subroutine sortorb(numfill,numorbs,neworbitals,neweng)
implicit none
include 'Include.blk'
integer i,j,k,m,n
double precision temp
*
do i = 1,2
  do j = 1,2
    do k = 1,numfill
      do m = numfill-1,1,-1
        if (neweng(m,i,j).gt.neweng(m+1,i,j)) then
          temp = neweng(m,i,j)
          neweng(m,i,j) = neweng(m+1,i,j)
          neweng(m+1,i,j) = temp
          do n = 1,numorbs
            temp = neworbitals(n,m,i,j)
            neworbitals(n,m,i,j) = neworbitals(n,m+1,i,j)
            neworbitals(n,m+1,i,j) = temp
          enddo
        endif
      enddo
    enddo
  enddo
enddo
*
return
end
*
*
*
subroutine writeorb(numorbs,localorb,root)
implicit none
include 'Include.blk'
double precision localorb(MAXORB,MAXORB,2,2)
character(*) root
character*(80) filename
*
integer i,j

```

```

*
  filename = root//'_grnd.orb_new'
  open (unit=17,file=filename,status='new',form='unformatted')
  filename = root//'_sing.orb_new'
  open (unit=18,file=filename,status='new',form='unformatted')
*
  write (17) numorbs,' UHF'
  do i = 1,numorbs
    do j = 1,numorbs
      write (17) localorb(i,j,1,1)
    enddo
    do j = 1,numorbs
      write (17) localorb(i,j,2,1)
    enddo
  enddo
*
  write (18) numorbs,' UHF'
  do i = 1,numorbs
    do j = 1,numorbs
      write (18) localorb(i,j,1,2)
    enddo
    do j = 1,numorbs
      write (18) localorb(i,j,2,2)
    enddo
  enddo
*
  close(17)
  return
end
*
* 1-D Integer
*
  subroutine zeroi1(array,m)
  implicit none
  integer i,m,array(m)
*
  do i = 1,m
    array(i) = 0
  enddo
*
  return
end
*
* 1-D Real
*
  subroutine zeror1(array,m)
  implicit none
  integer i,m

```

Appendix K: Program Listing of Corresp



```

double precision array(m)
*
do i = 1,m
  array(i) = 0.0d0
enddo
*
return
end
*
* 2-D Real
*
subroutine zeror2(array,m,n)
implicit none
integer i,j,m,n
double precision array(m,n)
*
do i = 1,m
  do j = 1,n
    array(i,j) = 0.0d0
  enddo
enddo
*
return
end
*
* 3-D Real
*
subroutine zeror3(array,m,n,o)
implicit none
integer i,j,k,m,n,o
double precision array(m,n,o)
*
do i = 1,m
  do j = 1,n
    do k = 1,o
      array(i,j,k) = 0.0d0
    enddo
  enddo
enddo
*
return
end
*
* 4-D Real
*
subroutine zeror4(array,m,n,o,p)
implicit none
integer i,j,k,l,m,n,o,p

```

Appendix K: Program Listing of Corresp

```

double precision array(m,n,o,p)
*
do i = 1,m
  do j = 1,n
    do k = 1,o
      do l = 1,p
        array(i,j,k,l) = 0.0d0
      enddo
    enddo
  enddo
enddo
*
return
end
*
*
*
subroutine zeroval(numfill,numorbs,usub,esub)
implicit none
include 'Include.blk'
*
integer i,j,numzero,new
double precision usub(MAXORB,MAXORB),esub(MAXORB),dot(MAXORB),ddot
double precision total
*
numzero = 0
do i = 1,numfill
  if (esub(i).lt.accura) then
    new = i
    numzero = numzero + 1
    do j = 1,numfill
      usub(j,i) = 1.0d0
    enddo
  endif
enddo
*
* If we have more than 1 zero eigenvalue we are in trouble
*
  if (numzero.gt.1) STOP"Too many zero eigenvalues"
*
* Calculate orthogonal matrix
*
do i = 1,numfill
  if(i.ne.new) dot(i) = ddot(numfill,usub(1,new),1,usub(1,i),1)
enddo
do i = 1,numfill
  if(i.ne.new)calldaxpy(numfill,-dot(i),usub(1,i),1,usub(1,new),1)
enddo

```

```

*
* Normalise the vector
*
      total = 0.0d0
      do i = 1,numorbs
        total = total + usub(i,new)**2
      enddo
      total = sqrt(total)
      do i = 1,numorbs
        usub(i,new) = usub(i,new) / total
      enddo
*
9980 format (5x,100f10.6)
*
      return
      end
*
* Common Block file
*
      integer MAXAT,MAXORB,numorbs,numfill,numat
      parameter (MAXAT=20,MAXORB=100)
*
*
*
      integer iatom(MAXORB),lfirst(MAXAT),llast(MAXAT)
      double precision charge(MAXAT,2),mopop(MAXORB,2),aopop(MAXORB,2)
*
* rhf declarations
*
* rhf:      ground & excited
* rhfoverlap: ground & excited
*           ground & excited
*
      double precisionrhf(MAXORB,MAXORB,2),rhfoverlap(MAXORB,MAXORB,2,2)
      double precision rhfeng(MAXORB,2)
*
* uhf declarations
*
*
* orbitals:  alpha   & beta
*           excited & ground
* The Rest:  alpha   & beta
*
      double precision orbitals(MAXORB,MAXORB,2,2)
      double precision D(MAXORB,MAXORB,2),DD(MAXORB,MAXORB,2)
      double precision V(MAXORB,MAXORB,2),L(MAXORB,MAXORB,2)
      double precision U(MAXORB,MAXORB,2),E(MAXORB,2)
      double precision energy(MAXORB,2,2),neweng(MAXORB,2,2)
*

```

Appendix K: Program Listing of Corresp


```
double precision neworbitals(MAXORB,MAXORB,2,2)
double precision newD(MAXORB,MAXORB,2)
logical gotzero(2),swap,mix,dump,char,outmol
double precision original(MAXORB,MAXORB,2,2),origeng(MAXORB,2,2)
*
double precision accura,lambda
data accura/1.0d-10/
```

Appendix L: Program Listing of Solid

```

      program solid
*
* Program to calculate the charges of the ground and excited states
* given my modified Mopac output of one of the states.
*
      implicit none
      include 'Params.inc'
      include 'Variables.inc'
*
      character*79 root
      integer len
*
* Read in the root filename
*
      call getargc(1,root,len)
*
* read in the value of Theta to be used
*
      call getargr(2,theta)
*
* read in the values of homo & ilumo
*
      call getargi(3,ihomo)
      call getargi(4,ilumo)
*
* zeros all arrays
*
      call clean
*
* Read in from the .bin file
*
      call read(root(1:len))
*
* Initialise some arrays and numbers
*
      call initarrays
*
* Say Hi
*
      call header(20,'Welcome to the SOLID program')
*
* Output all the data read in
*
      call start(20)

```

```

*
* calculate the fock matrix
*
      call calcfock (fockc,one,rjay,rkay,ihomo,norb)
*
      call printd1(fockc,norb,MAXORB,TITLE,NOLABEL,NOVERTICAL,
&                'Calculated Fock matrix (orbital energies)')
*
* calculate the energies
*
      call calceng
*
      call printnum('Ground energy',ground,NOMARK)
      call printnum('Singlet energy',singlet,NOMARK)
      call printnum('Doubly energy',doubly,NOMARK)
      call printnum('Energy between 0 & 2',eng02,NOMARK)
      call printnum('Energy between 1 & 2',eng12,NOMARK)
      write (20,*)
      call printnum('Singlet energy (by excitation)',singexd,NOMARK)
      call printnum('Doubly energy (by excitation)',doubexd,NOMARK)
*      call printnum('Energy 1 & 2 (by excitation)',eng12exd,NOMARK)
*
* End of reading data
*
      call header(20,'Gas Phase')
*
* Convert the rhf read in orbitals into uhf and print them out
*
      call rhf2uhf
*
      call printcoeff(orbitals(1,1,ALPH,GRND,GAS),norb,TITLE,LABEL,
& 'UHF Gas Phase Ground State orbitals (column vectors)')
*
      call printcoeff(orbitals(1,1,BETA,GRND,GAS),norb,TITLE,LABEL,
& 'UHF Gas Phase Ground State orbitals (beta) (column vectors)')
*
* Construct the mo populations and print them out
*
      call calcmopop(GAS)
*
      call printmoocc(mopop(1,1,GRND,GAS),norb,TITLE,LABEL,
& 'Gas Phase Ground State MO population')
*      call printmoocc(mopop(1,1,EXCD,GAS),norb,TITLE,LABEL,
& 'Gas Phase Excited State MO population')
*
      call calcaopop(GAS)
      call printaoocc(aopop(1,1,GRND,GAS),norb,TITLE,LABEL,
& 'Gas Phase Ground State AO population')

```

Appendix L: Program Listing of Solid

```

*      call printaoocc(aopop(1,1,EXCD,GAS),norb,TITLE,LABEL,
*      &                'Gas Phase Excited State AO population')
*
*      call calccharges(GAS)
*      call printchar(charges(1,1,GRND,GAS),nat,TITLE,LABEL,NOMARK,
*      &                'Gas Phase Ground State State Charges')
*      call printchar(charges(1,1,EXCD,GAS),nat,TITLE,LABEL,NOMARK,
*      &                'Gas Phase Excited State Charges')
*
* Now do the Solid phase
*
*      call header(20,'Solid Phase                               ')
*
* Convert the rhf read in orbitals into uhf and print them out
*
*      call calcsolid
*
*      call printcoeff(orbitals(1,1,ALPH,GRND,SOL),norb,TITLE,LABEL,
*      & 'UHF Solid Phase Ground State orbitals (column vectors)')
*
*      call printcoeff(orbitals(1,1,BETA,GRND,SOL),norb,TITLE,LABEL,
*      & 'UHF Solid Phase Ground State orbitals (beta) (column vectors)')
*
* Construct the mo populations and print them out
*
*      call calcmopop(SOL)
*
*      call printmoocc(mopop(1,1,GRND,SOL),norb,TITLE,LABEL,
*      &                'Solid Phase Ground State MO population')
*      call printmoocc(mopop(1,1,EXCD,SOL),norb,TITLE,LABEL,
*      &                'Solid Phase Excited State MO population')
*
*      call calcaopop(SOL)
*      call printaoocc(aopop(1,1,GRND,SOL),norb,TITLE,LABEL,
*      &                'Solid Phase Ground State AO population')
*      call printaoocc(aopop(1,1,EXCD,SOL),norb,TITLE,LABEL,
*      &                'Solid Phase Excited State AO population')
*
*      call calccharges(SOL)
*      call printchar(charges(1,1,GRND,SOL),nat,TITLE,NOLABEL,NOMARK,
*      &                'Solid Phase Ground State State Charges')
*
* Finally the Energy
*
*      write (20,*)
*      call printrnum('Theta (radians)                               ',theta,NOMARK)
*      call printrnum('Theta (degrees)                             ',theta/pi*180.0d0
*      &                ,NOMARK)

```

Appendix L: Program Listing of Solid

```

write (20,*)
call printnum('Energy of Solid Ground state ',solideng,NOMARK)
*
* Write out a .dat file suitable for Ewald input
*
call writeewald
*
* Bye bye
*
end
*
*
*
subroutine calcaopop(iphase)
*
* This subroutine calculate the ao populations
*
implicit none
include 'Params.inc'
include 'Variables.inc'
*
integer i,j,k,l,m,iphase
*
do i = 1,norb
do j = GRND,EXCD
aopop(i,COR,j,iphase) = 0.0d0
aopop(i,VAL,j,iphase) = 0.0d0
aopop(i,TOT,j,iphase) = 0.0d0
do l = 1,norb
do m = ALPH,BETA
if ((l.ne.ihomo).and.(l.ne.ilumo)) then
aopop(i,COR,j,iphase) = aopop(i,COR,j,iphase)
& + mopop(l,m,j,iphase) * orbitals(i,l,m,GRND,iphase) **2
endif
enddo
enddo
do l = ihomo,ilumo,ilumo-ihomo
do m = ALPH,BETA
aopop(i,VAL,j,iphase) = aopop(i,VAL,j,iphase)
& + mopop(l,m,j,iphase) * orbitals(i,l,m,GRND,iphase) **2
enddo
enddo
do l = 1,norb
do m = ALPH,BETA
aopop(i,TOT,j,iphase) = aopop(i,TOT,j,iphase)
& + mopop(l,m,j,iphase) * orbitals(i,l,m,GRND,iphase) **2
enddo
enddo
enddo

```

Appendix L: Program Listing of Solid

```

        enddo
    enddo
*
* Leave subroutine
*
    return
    end
*
*
*
    subroutine calccharges(iphase)
*
* turns the AO pops into charges
*
    implicit none
    include 'Params.inc'
    include 'Variables.inc'
*
    integer i,j,k,iphase
*
* Calculate the total, core & active charge
* From both the calculated MO's and the correct MO's
*
    do i = 1,norb
        do j = 1,MAXSPLIT
            do k = 1,MAXSTATE
                charges(iorb(i),j,k,iphase) = charges(iorb(i),j,k,iphase)
&                + aopop(i,j,k,iphase)
            enddo
        enddo
    enddo
*
* leave routine
*
    return
    end
*
*
*
    subroutine calceng
*
* Subroutine to calculate the energies
*
    implicit none
    include 'Params.inc'
    include 'Variables.inc'
*
    integer i,j

```

Appendix L: Program Listing of Solid

```

*
* Calculate the energy of the Ground state
*
    ground = 0.0d0
    do i = 1,nclose
        ground = ground + 2.0d0 * one(i,i)
        do j = 1,nclose
            ground = ground + 2.0d0 * rjay(i,j) - rkay(i,j)
        enddo
    enddo

*
* Calculate the Singlet state using only 1 & 2 electron integrals
*
    singlet = 0.0d0

*
    do i = 1,nclose
        if (i.ne.ihomo) singlet = singlet + 2.0d0 * one(i,i)
    enddo
    singlet = singlet + one(ihomo,ihomo) + one(ilumo,ilumo)

*
    do i = 1,nclose
        do j = 1,nclose
            if ((i.ne.ihomo).and.(j.ne.ihomo))
&        singlet = singlet + 2.0d0 * rjay(i,j) - rkay(i,j)
            enddo
        enddo

*
    do i = 1,nclose
        if (i.ne.ihomo) singlet = singlet + 2.0d0 * rjay(i,ihomo)
&    - rkay(i,ihomo) + 2.0d0 * rjay(i,ilumo) - rkay(i,ilumo)
    enddo

*
    singlet = singlet + rjay(ihomo,ilumo) + rkay(ihomo,ilumo)

*
* Calculate the Doubly excited Singlet using 1 & 2 electron integrals
*
    doubly = 0.0d0

*
    do i = 1,nclose
        if (i.ne.ihomo) doubly = doubly + 2.0d0 * one(i,i)
    enddo
    doubly = doubly + 2.0d0 * one(ilumo,ilumo)

*
    do i = 1,nclose
        do j = 1,nclose
            if ((i.ne.ihomo).and.(j.ne.ihomo))
&        doubly = doubly + 2.0d0 * rjay(i,j) - rkay(i,j)
            enddo
        enddo
    enddo

```

Appendix L: Program Listing of Solid

```

        enddo
*
        do i = 1,nclose
            if (i.ne.ihomo)
&      doubly = doubly + 4.0d0 * rjay(i,ilumo) - 2.0d0 * rkay(i,ilumo)
        enddo
*
        doubly = doubly + rjay(ilumo,ilumo)
*
        eng02 = rkay(ihomo,ilumo)
        eng12 = one(ihomo,ilumo) + triple(ihomo,ilumo,ilumo)
        do i = 1,norb
            if (i.ne.ihomo) eng12 = eng12 + 2.0d0 * triple(ihomo,ilumo,i)
        enddo
        eng12 = eng12 * sqrt(2.0d0)
*
* Now calculate the excitation energies - from p236 Szabo
* The rkay on the end of the singlet differs from Szabo cos theirs
* is a singlet determinant.
*
        singexd = ground + fock(ilumo) - fock(ihomo) - rjay(ilumo,ihomo)
&      + 2.0d0 * rkay(ilumo,ihomo)
*
        doubexd = ground + 2.0d0 * (fock(ilumo) - fock(ihomo))
&      + rjay(ihomo,ihomo) + rjay(ilumo,ilumo)
&      - 4.0d0 * rjay(ilumo,ihomo) + 2.0d0 * rkay(ilumo,ihomo)
*
* Leave the routine
*
        return
        end
*
*
*
        subroutine calcfock
*
* Subroutine to calculate the fock energy
*
        implicit none
        include 'Params.inc'
        include 'Variables.inc'
*
        integer i,j
*
        do i = 1,norb
            fockc(i) = one(i,i)
            do j = 1,nclose
                fockc(i) = fockc(i) + 2.0d0 * rjay(i,j) - rkay(i,j)
            enddo
        enddo

```

Appendix L: Program Listing of Solid


```

        enddo
    enddo
*
    return
end
*
*
*
    subroutine calcmopop(iphase)
*
* This sets up the molecular orbital occupancy using the number of
* orbitals and number of closed shells.
*
    implicit none
    include 'Params.inc'
    include 'Variables.inc'
*
    integer i,iphase
*
* set up the two MO occupancy arrays
*
    do i = 1,nclose
        mopop(i,ALPH,GRND,iphase) = 1.0d0
        mopop(i,BETA,GRND,iphase) = 1.0d0
        mopop(i,ALPH,EXCD,iphase) = 1.0d0
        mopop(i,BETA,EXCD,iphase) = 1.0d0
    enddo
*
    mopop(ihomo,ALPH,GRND,iphase) = 1.0d0
    mopop(ihomo,BETA,GRND,iphase) = 1.0d0
    mopop(ilumo,ALPH,GRND,iphase) = 0.0d0
    mopop(ilumo,BETA,GRND,iphase) = 0.0d0
*
    mopop(ihomo,ALPH,EXCD,iphase) = 1.0d0
    mopop(ihomo,BETA,EXCD,iphase) = 0.0d0
    mopop(ilumo,ALPH,EXCD,iphase) = 0.0d0
    mopop(ilumo,BETA,EXCD,iphase) = 1.0d0
*
*
* mopop(ihomo,ALPH,EXCD,GAS) = cos(theta)**2
* mopop(ilumo,ALPH,EXCD,GAS) = sin(theta)**2
* mopop(ihomo,BETA,EXCD,GAS) = cos(theta)**2
* mopop(ilumo,BETA,EXCD,GAS) = sin(theta)**2
*
    return
end
*
*
*

```

Appendix L: Program Listing of Solid

```

subroutine calcsolid
*
* This transforms the Gas phase orbitals into Solid phase orbitals
*
  implicit none
  include 'Params.inc'
  include 'Variables.inc'
*
  integer i,j
  double precision eng02,eng12
*
* j is the orbital variable
*
  do j = 1,norb
    if ((j.ne.ihomo).and.(j.ne.ilumo)) then
      do i = 1,norb
        orbitals(i,j,ALPH,GRND,SOL) = orbitals(i,j,ALPH,GRND,GAS)
        orbitals(i,j,BETA,GRND,SOL) = orbitals(i,j,BETA,GRND,GAS)
      enddo
    endif
  enddo
*
  do i = 1,norb
    orbitals(i,ihomo,ALPH,GRND,SOL) =
&    (cos(theta) * orbitals(i,ihomo,ALPH,GRND,GAS))
&    + (sin(theta) * orbitals(i,ilumo,ALPH,GRND,GAS))
    orbitals(i,ihomo,BETA,GRND,SOL) =
&    (cos(theta) * orbitals(i,ihomo,BETA,GRND,GAS))
&    + (sin(theta) * orbitals(i,ilumo,BETA,GRND,GAS))
    orbitals(i,ilumo,ALPH,GRND,SOL) =
&    - (sin(theta) * orbitals(i,ihomo,ALPH,GRND,GAS))
&    + (cos(theta) * orbitals(i,ilumo,ALPH,GRND,GAS))
    orbitals(i,ilumo,BETA,GRND,SOL) =
&    - (sin(theta) * orbitals(i,ihomo,BETA,GRND,GAS))
&    + (cos(theta) * orbitals(i,ilumo,BETA,GRND,GAS))
  enddo
*
* Calculate the solid ground state energy
*
  solideng = (cos(theta)**4 * ground) + (sin(theta)**4 * doubly)
&    + (2.0d0 * sin(theta)**2 * cos(theta)**2 * singlet)
&    + (2.0d0 * sin(theta)**2 * cos(theta)**2 * eng02)
&    + (sqrt(8.0d0) * sin(theta)**3 * cos(theta) * eng12)
*
  return
end
*
*

```

Appendix L: Program Listing of Solid

```

*
      subroutine clean
*
* This routine zeros or blanks all the arrays I am going to use.
* It calls the set of routine in zero.f
*
      implicit none
      include 'Params.inc'
      include 'Variables.inc'
*
      call zeroi1(ifirst,MAXAT)
      call zeroi1(ilast,MAXAT)
      call zeroi1(iatom,MAXAT)
      call zeroi1(iorb,MAXORB)
      call zeroc1(catom,MAXAT)
      call zerod2(rjay,MAXORB,MAXORB)
      call zerod2(rkay,MAXORB,MAXORB)
      call zerod2(orbssinit,MAXORB,MAXORB)
      call zerod4(mopop,MAXORB,MAXSPIN,MAXSTATE,MAXPHASE)
      call zerod5(orbitals,MAXORB,MAXORB,MAXSPIN,MAXSTATE,MAXPHASE)
      call zerod1(one,MAXORB,MAXORB)
      call zerod1(fock,MAXORB)
      call zerod1(fockc,MAXORB)
      call zerod1(nucchar,MAXAT)
      call zerod4(aopop,MAXORB,MAXSPLIT,MAXSTATE,MAXPHASE)
      call zerod4(charges,MAXAT,MAXSPLIT,MAXSTATE,MAXPHASE)
      call zerod3(triple,MAXORB,MAXORB,MAXORB)
*
      pi = acos (-1.0d0)
      theta = pi * theta / 180.d0
*
      return
      end
*
*
*
      subroutine getargc(n,param,ln)
*
* Routine to return a string off the command line and its length
* Uses the Input file structure that we have to use on the HP's
*
      implicit none
*
      character*(79) param
      integer i,n,ln
*
      open (unit=15,file='Input',status='old',form='formatted')
      do i = 1,n

```

```

        read (15,'(a79)') param
    enddo
    close (15)
*
    do i = 79,1,-1
        if (param(i:i).eq.' ') ln = i - 1
    enddo
*
    return
end
*
*
*
    subroutine getargr(n,param)
*
* Routine to return a real number off the command line
* Uses the Input file structure that we have to use on the HP's
*
    implicit none
*
    double precision param
    character*(79) line
    integer i,n,ln
*
    open (unit=15,file='Input',status='old',form='formatted')
    do i = 1,n
        read (15,'(a79)') line
    enddo
    read (line,*) param
    close (15)
*
    return
end
*
*
*
    subroutine getargi(n,param)
*
* Routine to return an integer number off the command line
* Uses the Input file structure that we have to use on the HP's
*
    implicit none
*
    integer param
    character*(79) line
    integer i,n,ln
*
    open (unit=15,file='Input',status='old',form='formatted')

```

Appendix L: Program Listing of Solid

```

do i = 1,n
  read (15,'(a79)') line
enddo
read (line,*) param
close (15)

*
  return
end
*
*
*
  subroutine header(line,string)
*
* routine that will print the input string to the line nicely
*
  implicit none
  include 'Params.inc'
*
  integer line,ln,i
  character*(28) string
*
  write (line,9990) '*****',
&'*****'
  write (line,9990) '*
&
  write (line,9980) '**',string,**
  write (line,9990) '*
&'
  write (line,9990) '*****',
&'*****'
  write (line,*)
  write (line,*)
*
9990 format (a40,a59)
9980 format (a1,1x,a28,68x,a1)
*
  return
end
*
*
*
  subroutine initarrays
*
* This routine sets up some usefull arrays and number based on the input
*
  implicit none
  include 'Params.inc'
  include 'Variables.inc'

```

```

*
      integer i,j
*
* Set up the array of which orbital belong to which atom
*
      do i = 1,norb
        do j = 1,nat
          if ((i.ge.ifirst(j)).and.(i.le.ilast(j))) then
            iorb(i) = j
          endif
        enddo
      enddo
*
* Sets up the nuclear charge of each inputted atom
*
      data atomlist/'h ','he','li','be','b ','c ','n ','o ','f ','ne'/
      data chargelist/1.0,2.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0/
      do i = 1,nat
        if (catom(i)(1:1).eq.' ') catom(i) = catom(i)(2:2)//' '
        do j = 1,ELEMENTS
          if (catom(i)(1:2).eq.atomlist(j)) nucchar(i) = chargelist(j)
        enddo
      enddo
*
* leave routine
*
      return
      end
*****
*
* This is a set of routines that will print out array of various type *
* with or without title and with or with labels. Subroutines such as *
* printd1 etc. are completely general and don't know about Params.inc. *
* Subroutines such as printcoeff are for precise array types. *
*
*****
      subroutine printd1 (array,filled,size,title_l,label_l,
        &                  vertical_l,string)
*
* This prints out a 1D array of doubles
* If vertical_l is false then label_l is ignored
*
      implicit none
*
      integer i,j,line,filled,size
      double precision array (size)
      character*(*) string
      logical label_l,title_l,vertical_l

```

Appendix L: Program Listing of Solid

```

*
    line = 20
*
    if (title_1) then
        write (line,9990) string
        write (line,*)
    endif
*
    if (vertical_1) then
        do i = 1,filled
            if (label_1)      write (line,9980) i,',',array(i)
            if (.not.label_1) write (line,9970)      array(i)
        enddo
    else
        write (line,9960) (array(i),i=1,filled)
    endif
*
    write (line,*)
    write (line,*)
*
    9990 format (a)
    9980 format (i5,a1,f11.5)
    9970 format (6x,f11.5)
    9960 format (6x,100f11.5)
*
    end
*****
    subroutine printd2(array,filled,size,title_1,label_1,string)
*
* This prints out a 2D square array of doubles
*
    implicit none
*
    integer i,j,line,filled,size
    double precision array(size,size)
    character(*) string
    logical label_1,title_1
*
    line = 20
*
    if (title_1) then
        write (line,9990) string
        write (line,*)
    endif
*
    if (label_1) then
        write (line,9960) ((' ',i,')',i=1,filled)
        write (line,*)

```

```

endif
*
do i = 1,filled
  if (label_1) write (line,9980) i,')',(array(i,j),j=1,filled)
  if (.not.label_1) write (line,9970) (array(i,j),j=1,filled)
enddo
write (line,*)
write (line,*)
*
9990 format (a)
9980 format (i5,a1,100f11.5)
9970 format (6x,100f11.5)
9960 format (5x,8(8x,a1,i1,a1))
*
end
*****
subroutine printd3(array,filled,size,title_1,label_1,string)
*
* This prints out a 3D square array of doubles
*
implicit none
*
integer i,j,k,line,filled,size
double precision array(size,size,size)
character*(*) string
logical label_1,title_1
*
line = 20
*
if (title_1) then
  write (line,9990) string
  write (line,*)
endif
do i = 1,filled
  do j = 1,filled
    if (label_1) write (line,9980) i,j,')',(array(i,j,k),k=1,filled)
    if (.not.label_1) write (line,9970) (array(i,j,k),k=1,filled)
  enddo
  write (line,*)
enddo
write (line,*)
write (line,*)
*
9990 format (a)
9980 format (2i5,a1,100f11.5)
9970 format (11x,100f11.5)
*
end

```

Appendix L: Program Listing of Solid


```

*****
      subroutine printcoeff(array,filled,title_l,label_l,string)
*
* This prints out a 2D square array of orbital coefficients
*
      implicit none
      include 'Params.inc'
*
      integer i,j,line,filled
      double precision array(MAXORB,MAXORB)
      character*(*) string
      logical label_l,title_l
*
      line = 20
*
      if (title_l) then
        write (line,9990) string
        write (line,*)
      endif
*
      if (label_l) then
        write (line,9960) ((' ',i,')',i=1,filled)
        write (line,*)
      endif
*
      do i = 1,filled
        write (line,9970)      (array(i,j),j=1,filled)
      enddo
      write (line,*)
      write (line,*)
*
      9990 format (a)
      9980 format (i5,a1,100f11.5)
      9970 format (6x,100f11.5)
      9960 format (5x,8(8x,a1,i1,a1))
*
      end
*****
      subroutine printmoocc(array,filled,title_l,label_l,string)
*
* This prints out 2 1D arrays of MO orbital occupancy (alpha & beta)
* The orbitals are printed in reverse order
*
      implicit none
      include 'Params.inc'
*
      integer i,j,line,filled
      double precision array(MAXORB,MAXSPIN)

```

```

character*(*) string
logical label_1,title_1
*
line = 20
*
if (title_1) then
  write (line,9990) string
  write (line,*)
endif
*
write (line,9990) '          Alpha      Beta'
write (line,*)
*
do i = filled,1,-1
  if (label_1) write (line,9980) i,')',(array(i,j),j=ALPH,BETA)
  if (.not.label_1) write (line,9970) (array(i,j),j=ALPH,BETA)
enddo
write (line,*)
write (line,*)
*
9990 format (a)
9980 format (i5,a1,100f11.5)
9970 format (6x,100f11.5)
*
end
*****
subroutine printnum(string,rnumber,mark_1)
*
* This prints out string and real number in a constant format
*
implicit none
include 'Params.inc'
*
integer line
double precision rnumber
character*(30) string
logical mark_1
*
line = 20
*
if (mark_1) then
  write (line,9990) '# ',string//blanks,rnumber
else
  write (line,9990) ' ',string//blanks,rnumber
endif
write (line,*)
*
9990 format (a2,a30,f15.5)

```

Appendix L: Program Listing of Solid

```

*
      end
*****
      subroutine printaoocc(array,filled,title_l,label_l,string)
*
* This prints out 3 1D arrays of AO orbital occupancy (COR, VAL & TOT)
*
      implicit none
      include 'Params.inc'
*
      integer i,j,line,filled
      double precision array(MAXORB,MAXSPLIT)
      character*(*) string
      logical label_l,title_l
*
      line = 20
*
      if (title_l) then
        write (line,9990) string
        write (line,*)
      endif
*
      write (line,9990) '          Core      Active      Total'
      write (line,*)
*
      do i = 1,filled
        if (label_l) write (line,9980) i,')',(array(i,j),j=1,MAXSPLIT)
        if (.not.label_l) write (line,9970) (array(i,j),j=1,MAXSPLIT)
      enddo
      write (line,*)
      write (line,*)
*
      9990 format (a)
      9980 format (i5,a1,4x,100f11.5)
      9970 format (10x,100f11.5)
*
      end
*****
      subroutine printchar(array,totat,title_l,label_l,mark_l,string)
*
* This prints out 3 1D arrays of charges (COR, VAL & TOT)
*
      implicit none
      include 'Params.inc'
      include 'Variables.inc'
*
      integer i,j,line,totat
      double precision array(MAXAT,MAXSPLIT),ch

```

Appendix L: Program Listing of Solid

```

character*(*) string
logical label_1,title_1,mark_1
*
line = 20
*
if (title_1) then
  write (line,9990) string
  write (line,*)
endif
*
write (line,9960) 'Core      Active      Total      Charge'
write (line,*)
*
do i = 1,totat
  ch = nucchar(i) - array(i,MAXSPLIT)
  if (label_1) then
    write (line,9980) i,')',catom(i),(array(i,j),j=1,MAXSPLIT),ch
  else if (mark_1) then
    write (line,9950) '#',i,')',catom(i),(array(i,j),j=1,MAXSPLIT),ch
  else
    write (line,9970) catom(i),(array(i,j),j=1,MAXSPLIT),ch
  endif
enddo
write (line,*)
write (line,*)
*
9990 format (a)
9980 format (i5,a1,1x,a2,1x,100f11.5)
9970 format (7x,a2,1x,100f11.5)
9960 format (16x,a)
9950 format (a1,i4,a1,1x,a2,1x,100f11.5)
*
end
*
*
*
*
subroutine read(name)
*
* Routine to read in all the data from the .orb file specified by the
* parameter. We initially assume that the file is ground state.
*
implicit none
include 'Params.inc'
include 'Variables.inc'
*
integer i,j,k,idum
character*(*) name
*
```

Appendix L: Program Listing of Solid

```

      open (unit=10,file=name//'.bin',status='old',form='unformatted')
*
      read (10) rhf_1
      read (10) nat
      read (10) norb
      read (10) nclose
      read (10) nopen
      read (10) nci
      read (10)
      read (10)
*
      do i =1,nat
        read (10) idum,ifirst(idum),ilast(idum),iatom(idum),catom(idum)
        if (idum.ne.i) then
          write (6,*) 'Mismatch in reading first & last from .bin file'
          STOP
        endif
      enddo
*
      do i = 1,norb
        read (10) (orbsinit(i,j),j=1,norb)
      enddo
*
      read (10) (fock(i),i=1,norb)
*
      do j =1,norb
        read (10) idum,(one(j,i),i=1,norb)
      enddo
*
      do j =1,norb
        read (10) idum,(rjay(j,i),i=1,norb)
      enddo
*
      do j =1,norb
        read (10) idum,(rkay(j,i),i=1,norb)
      enddo
*
      do i =1,norb
        do j =1,norb
          read (10) idum,idum,(triple(i,j,k),k=1,norb)
        enddo
      enddo
*
      close(10)
*
      return
      end
*
```

Appendix L: Program Listing of Solid

```

*
*
      subroutine rhf2uhf
*
* This routine will take the first array and split the rhf orbitals
* into alpha and beta - returned in the second array. Need to include
* Param.inc only so I know the dimensions of an orbital array
*
      implicit none
      include 'Params.inc'
      include 'Variables.inc'
*
      integer i,j
*
      do i = 1,norb
        do j = 1,norb
          orbitals(i,j,ALPH,GRND,GAS) = orbsinit(i,j)
          orbitals(i,j,BETA,GRND,GAS) = orbsinit(i,j)
        enddo
      enddo
*
      return
      end
*
*
*
      subroutine start(line)
*
* This subroutine prints out all the info we have initially read in, as
* well as any job control parameters that may be added later. All is
* printed to the line.
*
      implicit none
      include 'Params.inc'
      include 'Variables.inc'
*
      integer i,j,line
*
* This is explicitly printed out
*
      if (rhf_1)      write (line,9990) 'We are using RHF Orbitals'
      if (.not.rhf_1) write (line,9990) 'We are using UHF Orbitals'
      write (line,9980) 'Number of atoms is ',nat
      write (line,9980) 'Number of orbitals is ',norb
      write (line,9980) 'Number of closed orbitals is ',nclose
      write (line,9980) 'Number of open orbitals is ',nopen
      write (line,9980) 'Number of orbitals in CI is ',nci
      write (line,*)

```

Appendix L: Program Listing of Solid

```

write (line,9980) 'The Active Occupied MO is ',ihomo
write (line,9980) 'The Active Unoccupied MO is ',ilumo
write (line,*)
*
* This is explicitly printed out
*
  write (line,9990) 'For each atom, the first and last orbital'//
&      ', the element number, label & nuclear charge'
  do i = 1,nat
    write (line,9970) i,')',ifirst(i),ilast(i),iatom(i),catom(i),
& nucchar(i)
    enddo
  write (line,*)
*
  write (line,9990) 'For each orbital, the atom on which it sits'
  do i = 1,norb
    write (line,9970) i,')',iorb(i)
  enddo
  write (line,*)
*
*
*
  call printcoeff( orbsinit,norb,TITLE,LABEL,
&                'Orbital co-effs (column vectors)')
*
  call printd1(fock,norb,MAXORB,TITLE,LABEL,NOVERTICAL,
&             'Fock matrix (orbital energies)')
*
  call printd2(one,norb,MAXORB,TITLE,LABEL,
&             'one-electron integrals')
*
  call printd2(rjay,norb,MAXORB,TITLE,LABEL,
&             'two-electron j-integrals')
*
  call printd2(rkay,norb,MAXORB,TITLE,LABEL,
&             'two-electron k-integrals')
*
*   call printd3(triple,norb,MAXORB,TITLE,LABEL,'triple ( su | kk)')
*
*
9990 format (a)
9980 format (a,i5)
9970 format (i5,a,3i5,3x,a2,f10.5)
*
  end
*
*
*

```

```

      subroutine writeewald
*
* Subroutine to write the Ewald input .dat file
*
      implicit none
      include 'Params.inc'
      include 'Variables.inc'
*
      write (30,9990) 'title Unit Cell for LiF'
      write (30,9990) 'length angstroms 4.0173 4.0173 4.0173'
      write (30,9990) 'angles degrees 90.0 90.0 90.0'
      write (30,9990) 'ions 2'
      write (30,9980) 1,'Li',nucchar(1) - charges(1,MAXSPLIT,GRND,SOL)
      write (30,9980) 2,'F ',nucchar(2) - charges(2,MAXSPLIT,GRND,SOL)
      write (30,9990) 'atoms fractional 8'
      write (30,9990) '      0.0      0.0      0.0      1'
      write (30,9990) '      0.5      0.5      0.0      1'
      write (30,9990) '      0.0      0.5      0.5      1'
      write (30,9990) '      0.5      0.0      0.5      1'
      write (30,9990) '      0.5      0.5      0.5      2'
      write (30,9990) '      0.5      0.0      0.0      2'
      write (30,9990) '      0.0      0.5      0.0      2'
      write (30,9990) '      0.0      0.0      0.5      2'
      write (30,9990) 'accuracy 1.0d-5'
      write (30,9990) 'molecule 4'
      write (30,9990) 'end'
*
      9990 format (a)
      9980 format (i6,7x,a2,f14.5)
*
      return
      end
*****
*
* A set of subroutines that will zero, or blank, arrays of all sizes
*
*****
      subroutine zeroi1(array,m)
*
* Zeros a 1D array of integers
*
      implicit none
      integer i,m,array(m)
*
      do i = 1,m
         array(i) = 0
      enddo
*

```



```

        return
    end
*****
    subroutine zerod1(array,m)
*
* Zeros a 1D array of doubles
*
    implicit none
    integer i,m
    double precision array(m)
*
    do i = 1,m
        array(i) = 0.0d0
    enddo
*
    return
    end
*****
    subroutine zerod2(array,m,n)
*
* Zeros a 2D array of doubles
*
    implicit none
    integer i,j,m,n
    double precision array(m,n)
*
    do i = 1,m
        do j = 1,n
            array(i,j) = 0.0d0
        enddo
    enddo
*
    return
    end
*****
    subroutine zerod3(array,m,n,o)
*
* Zeros a 3D array of doubles
*
    implicit none
    integer i,j,k,m,n,o
    double precision array(m,n,o)
*
    do i = 1,m
        do j = 1,n
            do k = 1,o
                array(i,j,k) = 0.0d0
            enddo
        enddo
    enddo

```

```

        enddo
    enddo
*
    return
end
*****
    subroutine zerod4(array,m,n,o,p)
*
* Zeros a 4D array of doubles
*
    implicit none
    integer i,j,k,l,m,n,o,p
    double precision array(m,n,o,p)
*
    do i = 1,m
        do j = 1,n
            do k = 1,o
                do l = 1,p
                    array(i,j,k,l) = 0.0d0
                enddo
            enddo
        enddo
    enddo
*
    return
end
*****
    subroutine zerod5(array,m,n,o,p,q)
*
* Zeros a 5D array of doubles
*
    implicit none
    integer i,j,k,l,r,m,n,o,p,q
    double precision array(m,n,o,p,q)
*
    do i = 1,m
        do j = 1,n
            do k = 1,o
                do l = 1,p
                    do r = 1,q
                        array(i,j,k,l,r) = 0.0d0
                    enddo
                enddo
            enddo
        enddo
    enddo
*
    return

```

```

end
*****
      subroutine zeroc1(array,m)
*
* Blanks a 1D array of strings
*
      implicit none
      integer i,m
      character*(*) array(m)
*
      do i = 1,m
         array(i) = ''
      enddo
*
      return
      end
*****
*
* This include file contains my parameters and all the things I need
* to use my utilities and bits so I don't get confused. As it does
* not involve any actual parameters there are no common blocks.
*
*****
* First of all the parameters
*
*****
      integer MAXAT,MAXORB,MAXSTATE,MAXSPIN,MAXSPLIT,ELEMENTS,MAXPHASE
      parameter (MAXAT = 20, MAXORB = 100, MAXSTATE = 2, MAXSPIN = 2)
      parameter (MAXSPLIT = 3, ELEMENTS = 10, MAXPHASE = 2)
*****
*
* The following are useful functions
*
*****
      integer lnbink
*****
*
* The next little bit sets up useful values so I don't get confused
*
*****
      logical TITLE,NOTITLE,LABEL,NOLABEL,VERTICAL,NOVERTICAL
      logical MARK,NOMARK
      parameter (TITLE = .true., NOTITLE = .false.)
      parameter (LABEL = .true., NOLABEL = .false.)
      parameter (VERTICAL = .true., NOVERTICAL = .false.)
      parameter (MARK = .true., NOMARK = .false.)
*

```

Appendix L: Program Listing of Solid

```

integer ALPH,BETA,GRND,EXCD
parameter (ALPH = 1, BETA = 2, GRND = 1, EXCD = 2)
*
integer COR,VAL,TOT
parameter (COR = 1, VAL = 2, TOT = 3)
*
integer GAS,SOL
parameter (GAS = 1, SOL = 2)
*****
*
* Some fundamental constants
*
*****
double precision pi
common /fund/ pi
*****
*
* Intrinsic functions
*
*****
double precision cos,sin,acos
*****
*
* blank line
*
*****
character*(79) blanks
parameter(blanks = '
&
'//
')
*****
*
* This include file contains all the global variables. All of them
* should be in common blocks. Params.inc should always be used also
*
*****
*
* Logical from .bin file
*
logical rhf_1
common /readlog/ rhf_1
*
* Numbers from .bin file
*
integer nat,norb,nclose,nopen,nci,ihomo,ilumo
common /readnum/ nat,norb,nclose,nopen,nci,ihomo,ilumo
*
* Arrays holding the position of the first and last orbital on each
* numbered atom, the element number and atom label

```

Appendix L: Program Listing of Solid

```

* iorb holds the atom on which each orbital sits
*
      integer ifirst(MAXAT),ilast(MAXAT),iatom(MAXAT),iorb(MAXORB)
      character*2 catom(MAXAT)
      common /atominfo/ ifirst,ilast,iatom,catom,iorb
*
* Arrays holding the 1 & 2 electron integrals, the orbital energy and
* the orbital coefficients. All read in.
* 29/9/95 Added triple.
*
      double precision rjay(MAXORB,MAXORB),rkay(MAXORB,MAXORB)
      double precision orbsinit(MAXORB,MAXORB),one(MAXORB,MAXORB)
      double precision triple(MAXORB,MAXORB,MAXORB),fock(MAXORB)
      common /mopacorbs/ one,rjay,rkay,fock,orbsinit,triple
*
* Arrays that I calculate
*
      double precision fockc(MAXORB)
      common /malcorbs/ fockc
*
* The uhf orbitals and MO & AO occupancy
* 30/9/95 and now for both phases
*
      double precision orbitals(MAXORB,MAXORB,MAXSPIN,MAXSTATE,MAXPHASE)
      double precision mopop(MAXORB,MAXSPIN,MAXSTATE,MAXPHASE)
      double precision aopop(MAXORB,MAXSPLIT,MAXSTATE,MAXPHASE)
      common /uhforbs/ orbitals,mopop,aopop
*
* energy variables
*
      double precision ground,singlet,doubly,eng02,eng12,solideng
      double precision singexd,doubexd,eng02exd,eng12exd
      common /energy/ ground,singlet,doubly,eng02,eng12,solideng,
& singexd,doubexd,eng02exd,eng12exd
*
* Charges
*
      double precision charges(MAXAT,MAXSPLIT,MAXSTATE,MAXPHASE)
      double precision chargelist(ELEMENTS),nucchar(MAXAT)
      character*2 atomlist(ELEMENTS)
      common /charges/ charges,nucchar,chargelist,atomlist
*
* Theta
*
      double precision theta
      common /theta/ theta

```

Appendix M: Solid Output for LiF

```
*****
*
* Welcome to the SOLID program
*
*****
```

```
We are using RHF Orbitals
Number of atoms is      2
Number of orbitals is   8
Number of closed orbitals is  4
Number of open orbitals is  0
Number of orbitals in CI is  2
```

```
The Active Occupied MO is      1
The Active Unoccupied MO is    8
```

For each atom, the first and last orbital, the element number, label & nuclear charge

```
1)   1   4   3   li   1.00000
2)   5   8   9   f    7.00000
```

For each orbital, the atom on which it sits

```
1)   1
2)   1
3)   1
4)   1
5)   2
6)   2
7)   2
8)   2
```

Orbital co-effs (column vectors)

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
.14884	-.15960	.00000	.00000	-.87303	.00000	.00000	-.43611
.21924	-.15254	.00000	.00000	.48209	.00000	.00000	-.83442
.00000	.00000	-.17173	.00355	.00000	-.98493	-.02035	.00000
.00000	.00000	.00355	.17173	.00000	.02035	-.98493	.00000
.96312	.10632	.00000	.00000	.02177	.00000	.00000	.24620
-.04662	.96951	.00000	.00000	-.07025	.00000	.00000	-.23008
.00000	.00000	-.98493	.02035	.00000	.17173	.00355	.00000
.00000	.00000	.02035	.98493	.00000	-.00355	.17173	.00000

Fock matrix (orbital energies)

-35.83281 -10.32503 -10.10889 -10.10889 .03437 2.40056 2.40056 5.80529

one-electron integrals

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
-135.65305	-3.04604	.00000	.00000	-.39346	.00000	.00000	-12.56353
-3.04604	-109.75599	.00000	.00000	3.54891	.00000	.00000	9.29519
.00000	.00000	-109.85947	.00000	.00000	11.08280	.00008	.00000
.00000	.00000	.00000	-109.85947	.00000	.00008	-11.08280	.00000
-.39346	3.54891	.00000	.00000	-35.79604	.00000	.00000	-7.54312
.00000	.00000	11.08280	.00008	.00000	-41.44875	.00000	.00000
.00000	.00000	.00008	-11.08280	.00000	.00000	-41.44875	.00000
-12.56353	9.29519	.00000	.00000	-7.54312	.00000	.00000	-60.54025

two-electron j-integrals

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
15.73639	16.06217	16.24387	16.24387	4.55934	5.62432	5.62432	8.71600
16.06217	16.07260	14.31232	14.31232	4.49908	5.49161	5.49161	8.40955
16.24387	14.31232	16.04405	14.34742	4.47177	5.57902	5.49962	8.44589
16.24387	14.31232	14.34742	16.04405	4.47177	5.49962	5.57902	8.44589
4.55934	4.49908	4.47177	4.47177	7.06993	5.18916	5.18916	5.06889
5.62432	5.49161	5.57902	5.49962	5.18916	5.02500	4.57048	4.94051
5.62432	5.49161	5.49962	5.57902	5.18916	4.57048	5.02500	4.94051
8.71600	8.40955	8.44589	8.44589	5.06889	4.94051	4.94051	6.52704

two-electron k-integrals

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
15.73639	4.26325	4.37635	4.37635	.05165	.13586	.13586	.63131
4.26325	16.07260	.87601	.87601	.07011	.05128	.05128	.42930
4.37635	.87601	16.04405	.84832	.02589	.32048	.03221	.31427
4.37635	.87601	.84832	16.04405	.02589	.03221	.32048	.31427
.05165	.07011	.02589	.02589	7.06993	.66923	.66923	.47641
.13586	.05128	.32048	.03221	.66923	5.02500	.22726	.35111
.13586	.05128	.03221	.32048	.66923	.22726	5.02500	.35111
.63131	.42930	.31427	.31427	.47641	.35111	.35111	6.52704

Calculated Fock matrix (orbital energies)

-35.83281 -10.32503 -10.10889 -10.10889 .03437 2.40056 2.40056 5.80529

Ground energy -531.50362

Singlet energy -497.31890

Appendix M: Solid Output for LiF

Doubly energy -459.56536

Energy between 0 & 2 .63131

Energy between 1 & 2 6.30141

Singlet energy (by excitation) -497.31890

Doubly energy (by excitation) -459.56536

*

* Gas Phase *

*

UHF Gas Phase Ground State orbitals (column vectors)

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
.14884	-.15960	.00000	.00000	-.87303	.00000	.00000	-.43611
.21924	-.15254	.00000	.00000	.48209	.00000	.00000	-.83442
.00000	.00000	-.17173	.00355	.00000	-.98493	-.02035	.00000
.00000	.00000	.00355	.17173	.00000	.02035	-.98493	.00000
.96312	.10632	.00000	.00000	.02177	.00000	.00000	.24620
-.04662	.96951	.00000	.00000	-.07025	.00000	.00000	-.23008
.00000	.00000	-.98493	.02035	.00000	.17173	.00355	.00000
.00000	.00000	.02035	.98493	.00000	-.00355	.17173	.00000

Gas Phase Ground State MO population

	Alpha	Beta
8)	.00000	.00000
7)	.00000	.00000
6)	.00000	.00000
5)	.00000	.00000
4)	1.00000	1.00000
3)	1.00000	1.00000
2)	1.00000	1.00000
1)	1.00000	1.00000

Gas Phase Ground State AO population

	Core	Active	Total
1)	.05094	.04431	.09525
2)	.04654	.09613	.14267
3)	.05900	.00000	.05900

4)	.05900	.00000	.05900
5)	.02261	1.85521	1.87782
6)	1.87991	.00435	1.88426
7)	1.94100	.00000	1.94100
8)	1.94100	.00000	1.94100

Gas Phase Ground State State Charges

	Core	Active	Total	Charge
1) li	.21549	.14044	.35593	.64407
2) f	5.78451	1.85956	7.64407	-.64407

```
*****
*
* Solid Phase
*
*****
```

UHF Solid Phase Ground State orbitals (column vectors)

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
.11026	-.15960	.00000	.00000	-.87303	.00000	.00000	-.44743
.14568	-.15254	.00000	.00000	.48209	.00000	.00000	-.85035
.00000	.00000	-.17173	.00355	.00000	-.98493	-.02035	.00000
.00000	.00000	.00355	.17173	.00000	.02035	-.98493	.00000
.98092	.10632	.00000	.00000	.02177	.00000	.00000	.16132
-.06650	.96951	.00000	.00000	-.07025	.00000	.00000	-.22514
.00000	.00000	-.98493	.02035	.00000	.17173	.00355	.00000
.00000	.00000	.02035	.98493	.00000	-.00355	.17173	.00000

Solid Phase Ground State MO population

	Alpha	Beta
8)	.00000	.00000
7)	.00000	.00000
6)	.00000	.00000
5)	.00000	.00000
4)	1.00000	1.00000
3)	1.00000	1.00000
2)	1.00000	1.00000
1)	1.00000	1.00000

Solid Phase Ground State AO population

Core	Active	Total
------	--------	-------

Appendix M: Solid Output for LiF

1)	.05094	.02432	.07526
2)	.04654	.04245	.08899
3)	.05900	.00000	.05900
4)	.05900	.00000	.05900
5)	.02261	1.92439	1.94700
6)	1.87991	.00884	1.88876
7)	1.94100	.00000	1.94100
8)	1.94100	.00000	1.94100

Solid Phase Ground State State Charges

	Core	Active	Total	Charge
li	.21549	.06676	.28225	.71775
f	5.78451	1.93324	7.71775	-.71775

Theta (radians) .08727

Theta (degrees) 5.00000

Energy of Solid Ground state -530.96279

12: References

- ¹ R. S. Mulliken, *J. Chem. Phys.*, **23**, 1833 (1955).
- ² P. Politzer, K. C. Leung, J. D. Elliot, S. K. Peters, *Theor. Chim. Acta*, **38**, 101 (1975).
- ³ A. Julg, *Topics in Current Chemistry*, **58**, 1 (1975).
- ⁴ R. F. W. Bader, R. M. Beddell, P. E. Cade, *J. Am. Chem. Soc.*, **93**, 3095 (1971).
- ⁵ K. Jug, *Theor. Chim. Acta*, **29**, 29 (1973).
- ⁶ L. E. Chirlian, M. M. Francl, *J. Comp. Chem.*, **8**, 894 (1987).
- ⁷ M. Orozco, F. J. Luque, *J. Comp. Chem.*, **11**, 909 (1990).
- ⁸ G. Tasi, I. Kiricsi, H. Förster, *J. Comp. Chem.*, **13**, 371 (1992).
- ⁹ J. Rodriguez, F. Manuat, F. Sanz, *J. Comp. Chem.*, **14**, 922 (1993).
- ¹⁰ Z. Su, *J. Comp. Chem.*, **14**, 1036 (1993).
- ¹¹ C. Alemán, F. J. Luque, M. Orozco, *J. Comp. Chem.*, **14**, 799 (1993).
- ¹² F. A. Momany, *J. Phys. Chem.*, **82**, 592 (1978).
- ¹³ R. Dovesi, M. Causà, R. Orlando, C. Roetti, V. R. Saunders, *J. Chem. Phys.*, **92**, 7402 (1990).
- ¹⁴ V. Vitek, D. J. Srolovitz, *Atomistic Simulation of Materials*, Plenum, New York (1989).
- ¹⁵ E. A. Colbourn, *Advances in Solid Structure Chemistry* (ed. C. R. A. Catlow), **1**, 1 (1989).
- ¹⁶ N. F. Mott, M. J. Littleton, *Trans. Faraday Soc.*, **34**, 485 (1938);
C. R. A. Catlow, *J. Chem. Soc., Faraday Trans. 2*, **85**, 335 (1989);
A. B. Lidiard, *J. Chem. Soc., Faraday Trans. 2*, **85**, 341 (1989).
- ¹⁷ C. Pisani, R. Dovesi, C. Roetti, *Lecture notes in Chemistry*, Vol. 48, Springer, Berlin (1988); R. Dovesi, V. R. Saunders, C. Roetti, *CRYSTAL2. User Documentation*, University of Torino and SERC Daresbury Laboratory.
- ¹⁸ M. J. Norgett, UKAEA Report, AERE-R.7650 (1974); C. R. A. Catlow, R. James, W. C. Mackrodt, R. F. Stewart, *Phys. Rev.* **B25**, 1006 (1982);
C. R. A. Catlow, M. J. Norgett, UKAEA Report AERE-M.2763 (1978).
- ¹⁹ A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction*

- to *Advanced Electronic Structure Theory*, 1982, Macmillan Publishing Co., Inc.
- ²⁰ p.68, A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, 1982, Macmillan Publishing Co., Inc.
- ²¹ p.135, A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, 1982, Macmillan Publishing Co., Inc.
- ²² p.134, A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, 1982, Macmillan Publishing Co., Inc.
- ²³ p.236, A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, 1982, Macmillan Publishing Co., Inc.
- ²⁴ p.129, A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, 1982, Macmillan Publishing Co., Inc.
- ²⁵ p.70, A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, 1982, Macmillan Publishing Co., Inc.
- ²⁶ p.157, A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, 1982, Macmillan Publishing Co., Inc.
- ²⁷ P. Ewald, *Ann. Phys.*, **64**, 253 (1921).
- ²⁸ Appendix B: Ewald calculation of lattice sums, p.604, C. Kittel, *Introduction to Solid State Physics*, 1986, Wiley, Sixth Edition.
- ²⁹ C. R. A. Catlow, M. J. Norgett, UKAEA Report AERE-M.2939.
- ³⁰ M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, 1987, Oxford Science Publications.
- ³¹ p.315, Coulson, *Valence*, 1961, Oxford University Press, Second Edition.
- ³² p.596, P. W. Atkins, *Physical Chemistry*, 1986, Oxford University Press, 3rd Edition.
- ³³ F-130, R. C. Weast, *Handbook of Chemistry and Physics*, 1987, CRC Press, Inc., 1st Student Edition.

- ³⁴ F. S. Gallaso, *Structure and Properties of Inorganic Solids*, 1970, Pergamon Press Ltd.
- ³⁵ p.63, F. S. Gallaso, *Structure and Properties of Inorganic Solids*, 1970, Pergamon Press Ltd.
- ³⁶ p.42, F. S. Gallaso, *Structure and Properties of Inorganic Solids*, 1970, Pergamon Press Ltd.
- ³⁷ p.123, F. S. Gallaso, *Structure and Properties of Inorganic Solids*, 1970, Pergamon Press Ltd..
- ³⁸ p.92, F. S. Gallaso, *Structure and Properties of Inorganic Solids*, 1970, Pergamon Press Ltd..
- ³⁹ p.46, F. S. Gallaso, *Structure and Properties of Inorganic Solids*, 1970, Pergamon Press Ltd..
- ⁴⁰ p.892, W. J. Moore, *Physical Chemistry*, Longman, London (1987), Fifth Edition.
- ⁴¹ M. J. S. Dewar, W. Thiel, J. Am. Chem. Soc., **99**, 4899 (1977).
- ⁴² M. J. S. Dewar, E. G. Zeobisch, E. F. Healy, J. J. P. Stewart, J. Am. Chem. Soc., **97**, 1294 (1975).
- ⁴³ J. J. P. Stewart, J. Comp. Chem., **10**, 209 (1989); J. J. P. Stewart, J. Am. Chem. Soc., **10**, 221 (1987).
- ⁴⁴ MOPAC, Version 5.00 Manual, available from QCPE, University of Indiana, Bloomington, IN 47406; J.J.P.Stewart, J. Computer-aided Molecular Design, **4**(1), 1 (1990).
- ⁴⁵ K. P. Huber and G. Herzberg, *Constants of Diatomic Molecules*, 1979, Van Nostrand Reinhold.
- ⁴⁶ B. Bak, D. Christensen, W. B. Dixon, L. Hansen-Nygaard, J. R. Andersen, M. Schottländer, J. Mol. Spectrosc., **9**, 124 (1962).
- ⁴⁷ U. Meier, V. Staemmler, Theor. Chim. Acta, **76**, 95 (1989).
- ⁴⁸ F-107, R. C. Weast, *Handbook of Chemistry and Physics*, 1987, CRC Press, Inc., 1st Student Edition.
- ⁴⁹ F-110, R. C. Weast, *Handbook of Chemistry and Physics*, 1987, CRC Press, Inc., 1st Student Edition.
- ⁵⁰ W. D. Laidig, P. Saxe, H. F. Schaefer III, J. Chem. Phys., **73**, 1765 (1980).
- ⁵¹ E. S. Nielsen, P. Jørgensen, J. Oddershede, J. Chem. Phys., **73**, 6238

(1980).

⁵² K-H. Thunemann, R. J. Buenker, W. Butscher, J. Chem. Phys., **47**, 313 (1980).

⁵³ H. Nakatsuji, K. Hirao, Y. Mizukami, Chem. Phys. Lett., **179**, 555 (1991).

⁵⁴ H. F. King, R. E. Stanton, H. Kim, R. E. Wyatt, R. G. Parr, J. Chem. Phys., **47**, 1936 (1975); A. T. Amos, G. G. Hall, Proc. Roy. Soc. (London), **A263**, 483 (1961); T. L. Gilbert, J. Chem. Phys., **43**, S248 (1965).

⁵⁵ P. B. Karadakov, J. Garrett, D. L. Cooper, M. Raimondi, J. Chem. Soc. Faraday Trans., **90**, 1643 (1994).

⁵⁶ K. H. Homann, Abbreviated list of Quantities, Units and Symbols in Physical Chemistry, 1987, Blackwell Scientific Publications.

13: Tables

- p.20 **Table 1** Time taken to calculate the total energy for various values of ν
- p.22 **Table 2** Comparison of Lattice Energies
- p.27 **Table 3** Systems Investigated
- p.27 **Table 4** Excitation Energies in the literature (in eV)
- p.28 **Table 5** Excitation Energies Calculated by **MOPAC** (in eV)
- p.40 **Table 6** Energies and charges for various orbital combinations
- p.44 **Table 7** Atomic units
- p.44 **Table 8** Fundamental Constants

